Algoritmos de Escalonamento para Redução do Consumo de Energia em Computação em Nuvem

Pedro Paulo Vezzá Campos

Monografia apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Bacharel em Ciência da Computação

Programa: Bacharelado em Ciência da Computação

Orientador: Prof. Dr. Daniel Macêdo Batista

1 de dezembro de 2013

Agradecimentos

À minha família, pelo amor, paciência e apoio incondicionais.

Aos meus amigos todos, por terem tornado a vida acadêmica muito mais divertida e diversificada.

Ao professor Daniel Batista, pela sugestão de tema para o TCC, reuniões, críticas e comentários sempre pertinentes.

À Elaine Naomi Watanabe por todas as ideias, ajuda e tempo dedicados no apoio à produção dos resultados apresentados nesta monografia.

Às agências de fomento por viabilizarem a dedicação em tempo integral do autor às atividades acadêmicas.

Resumo

CAMPOS, P P. V. **Algoritmos de Escalonamento para Redução do Consumo de Energia em Computação em Nuvem**. 2013. 52 p. Monografia (Graduação) – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2013.

Com o contínuo barateamento de insumos computacionais tais como poder de processamento, armazenamento e largura de banda de rede, há uma tendência atual de migração de serviços para nuvens computacionais, capazes de processar grandes quantidades de dados (*Big data*) gerando resultados a um custo aceitável.

Um dos maiores custos envolvidos na operação de uma nuvem vem da energia necessária para manter o parque de servidores operando e refrigerado. Este trabalho de conclusão de curso apresenta dois algoritmos de escalonamento de tarefas para computação em nuvem e variantes que reduzem tal consumo energético. Seus desempenhos foram comparados com outros três algoritmos, um clássico e dois retirados da bibliografia da área.

Um simulador de computação em nuvem foi utilizado juntamente com cargas de trabalho realistas em experimentos que evidenciem o comportamento do algoritmo em diferentes condições de uso.

Palavras-chave: computação em nuvem, escalonamento, consumo energético, CloudSim, DAG.

Abstract

CAMPOS, P. P. V. **Scheduling Algorithms for Green Cloud Computing**. 2013. 52 p. Dissertation (Graduation) – Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2013.

With the continuous cheapening of computational resources such as processing power, storage and bandwidth, there is a current trend of migration of services to cloud providers, capable of processing large amounts of data (Big data) generating results at a reasonable price.

One of the biggest costs involved in the operation of a cloud is the energy necessary to keep the server pool operating and refrigerated. This work presents two task scheduling algorithms for cloud computing environments and variants that reduce such energy consumption. Their performances were compared with three other algorithms, one classic and two obtained from the area's bibliography.

A cloud computing simulator was used together with realistic workloads in experiments that highlight the algorithm's behavior in different usage conditions.

Keywords: cloud computing, task scheduling, energy consumption, CloudSim, DAG.

Only those who attempt the absurd will achieve the impossible.

M. C. Escher

Sumário

Lista de Abreviaturas

API	Interface para programação de aplicativo (*Application programming interface*)
CPU	Unidade central de processamento (*Central processing unit*)
DAG	Grafo acíclico dirigido (*Directed acyclic graph*)
DVFS	Dimensionamento dinâmico de tensão e frequência (*Dynamic voltage and frequency scaling*)
HEFT	Tempo heterogêneo mais cedo de conclusão (*Heterogeneous earliest finish time*)
TI	Tecnologia da informação
XML	Linguagem de marcação extensível (*eXtensible Markup Language*)

Lista de Figuras

Lista de Tabelas

Parte I

Parte Objetiva

Capítulo 1

Introdução

Esta monografia desenvolvida durante o ano de 2013 para a disciplina MAC0499 – Trabalho de Formatura Supervisionado apresenta os trabalhos realizados no estudo e experimentação de técnicas de escalonamento de tarefas em ambientes de computação em nuvem sob a orientação do professor Daniel Macêdo Batista.

Em conjunto com a aluna de mestrado Elaine Watanabe, foi desenvolvido e avaliado um novo algoritmo que fosse energeticamente eficiente. O escalonador deve atender aos requisitos da aplicação e ao mesmo tempo buscar uma alocação de recursos próxima da ótima em termos de economia de energia. Enquanto a aluna focou na concepção do algoritmo, o aluno dedicou-se a adaptar simuladores existentes para validar o algoritmo e realizar experimentos para estudar seu comportamento diante de diferentes cargas de trabalho.

A parte objetiva deste trabalho está organizada da seguinte forma: no Capítulo 2 são apresentados brevemente os conceitos que fundamentam pesquisas na área e que são necessários para a compreensão dos capítulos seguintes. O Capítulo 3 trata dos algoritmos propostos para esta monografia juntamente com as motivações e intuições empregadas em cada algoritmo. Posteriormente, no Capítulo 4 o foco é direcionado para os resultados experimentais obtidos em um simulado de computação em nuvem ao comparar os algoritmos desenvolvidos e variantes com outros retirados da bibliografia. Conclusões e considerações finais são descritas no Capítulo 5.

Já a parte subjetiva está organizada em dois tópicos principais: no Capítulo 6 há uma reflexão acerca do processo de produção deste trabalho e sua aplicação. Já no Capítulo 7 há uma reflexão mais ampla sobre as experiências vividas em cinco anos de graduação em duas universidades distintas e uma previsão dos próximos passos a seguir na vida profissional.

1.1 Motivação

Desde a década de 1970 a oferta de poder computacional, armazenamento e comunicação vem crescendo em um ritmo exponencial em função do tempo. Até o fim da década de 1990 essas necessidades vinham sendo supridas com aperfeiçoamentos nas arquiteturas dos computadores e melhorias no processo produtivo. A Lei de Moore continuava se mostrando válida, duplicando o poder dos computadores e servidores a cada 18 meses e, junto com esse aumento, impondo uma necessidade energética cada vez maior para manter o computador funcionando e refrigerado. Porém, nos anos 2000 percebeu-se que o projeto de processadores encontrou uma barreira de potência. Processadores da época, tal como o Pentium 4, dissipavam 100W de potência e sua eficiência energética era baixa. [PH12] Assim, surgiu uma nova tendência, processadores mais simples e mais paralelos utilizando novas técnicas de economia de energia. Em suma, surgiu uma demanda por uma computação mais "verde" (*green computing*), que valorizasse a sustentabilidade dos seus processos e a economia de recursos.

Iniciou-se, assim, uma tendência de concentração do poder de computação e armazenamento em torno dos grandes *data centers* e *data warehouses*. A computação em nuvem (*cloud computing*) passou a ser apresentada como uma solução para a redução de custos e desperdícios através da

racionalização de recursos computacionais. Na Seção 2.1 são apresentadas as inovações presentes nesse modelo de computação. Porém, o sucesso dessa metodologia depende de estratégias inteligentes que permitam gerenciar os recursos disponíveis a fim de realizar uma economia de escala sem descumprir os requisitos de qualidade dos usuários.

Em um nível mais técnico, uma nuvem é projetada para executar tarefas, estas subdivididas em subtarefas. Cada subtarefa pode possuir uma demanda específica de ambiente para ser executada, sistema operacional, programas instalados, poder mínimo de processamento, armazenamento, etc. Ainda, subtarefas podem depender de que uma subtarefa anterior tenha sido concluída antes de poder ser executada. Em [CBdF11] e [BCdF11] Chaves e Batista mostraram que é possível modelar tais tarefas como digrafos acíclicos (DAGs) que incorporem as demandas de ambiente. Ainda, desenvolveram uma heurística para escalonar as subtarefas em computação em grade, similar à computação em nuvem, visando diminuir o tempo de conclusão da tarefa através da redução do tráfego de rede.

1.2 Objetivos

Este trabalho tem por objetivo implementar e validar uma nova heurística para o problema apresentado que reduza o consumo energético sem grandes prejuízos ao tempo de execução da tarefa. A heurística foi desenvolvida em conjunto com Elaine Naomi Watanabe (elainew@ime.usp.br), aluna de Mestrado em Computação do IME/USP. O desempenho desse algoritmo é comparado com outros algoritmos com objetivos similares encontrados na literatura. Para isso será feito uso de um simulador de computação em nuvem, o CloudSim_DVFS, a ser detalhado na Seção 2.4.

1.3 Desafios

As dificuldades enfrentadas nesta monografia vem de duas fontes: a primeira é tecnológica, há diversos simuladores de computação em nuvem ou em grade, porém o tópico de simulação energética é relativamente recente como atividade de pesquisa. Sendo assim, de todos programas de simulação estudados, CloudSim, GridSim, SimGrid, WorkflowSim e CloudSim_DVFS, apenas o primeiro e o último possuem tal funcionalidade disponível para ser utilizada no momento. O simulador, WorkflowSim, apesar de ser baseado no CloudSim não tem por padrão a API de simulação energética disponível. Uma das tarefas da monografia foi justamente tentar resolver tal problema.

O outro desafio é uma questão computacional mais fundamental: o problema de escalonar tarefas em diversos processadores (Num sentido mais amplo do que seria um processador) é NP-difícil [Sin07]. Para contornar esse problema diversas heurísticas já foram propostas, inclusive a apresentada nesta monografia. Um trecho do livro *Task Scheduling for Parallel Systems* de Oliver Sinnen resume a relação custo-benefício que deve ser ponderada para a descoberta de um escalonamento ótimo:

> Unfortunately, finding a schedule of minimal length (i.e., an optimal schedule) is in general a difficult problem. This becomes intuitively clear as one realizes that an optimal schedule is a trade-off between high parallelism and low interprocessor communication. On the one hand, nodes should be distributed among the processors in order to balance the workload. On the other hand, the more the nodes are distributed, the more interprocessor communications, which are expensive, are performed. In fact, the general decision problem (a decision problem is one whose answer is either “yes” or “no”) associated with the scheduling problem is NP-complete. [Sin07]

Capítulo 2

Conceitos

2.1 Computação em Nuvem

Computação em nuvem é uma expressão utilizada para definir o fornecimento e uso de insumos de computação como um serviço. Apesar do termo ainda não possuir uma definição precisa, o conceito fundamental é que computação em nuvem pressupõe um serviço[1] elástico, virtualmente ilimitado e pago apenas pela porção realmente utilizada, muito similar ao sistema de distribuição elétrica [AFG+09]. Computação em nuvem tornou-se um negócio atrativo a fornecedores e clientes graças ao barateamento de insumos necessários à computação, como energia, poder de processamento, armazenamento e transmissão de dados, permitindo uma economia de escala.

O objetivo final da computação em nuvem é prover um serviço ubíquo ao usuário, empresas ou pessoas físicas, que delegariam a gestão dessa informação a terceiros competentes para prover um serviço de qualidade e seguro. Grandes empresas da área de tecnologia possuem soluções de computação em nuvem, dentre as quais podemos citar Amazon [2], Google [3], Microsoft [4] e IBM [5].

Uma importante vantagem de *cloud computing* é que com essa concentração de dados e serviços é possível desenvolver técnicas de otimização do uso de grandes *data centers*. Segundo estudo realizado por Barroso e Hölzle [BH07] em 5000 servidores do Google, raramente eles permanecem completamente ociosos e dificilmente operam próximos da sua utilização máxima. Na maior parte do tempo estão trabalhando entre 10% e 50% do nível máximo. Os autores mostram que justamente nessa faixa de utilização tais servidores são menos eficientes energeticamente.

Computação em nuvem é uma candidata a ajudar a melhorar essa perspectiva. Através de virtualização e reposicionamento automático de máquinas virtuais no data center, uma funcionalidade disponível em produtos pagos como o VMware vSphere [VMw] e softwares livres como o Xen [AU09], é possível dimensionar qual parcela do data center estará ativa em um dado momento dependendo da demanda. Servidores com pouca utilização podem ser virtualizados em um único servidor físico de modo que este trabalhe com uma utilização que seja mais eficiente.

Vale ressaltar que em uma situação ideal, toda essa consolidação de servidores é transparente ao usuário final. Em caso de um pico na demanda por um determinado serviço, o provedor da nuvem deve garantir que haja uma resposta rápida da infraestrutura para suportar a nova carga requisitada. Dessa forma, são respeitados os acordos de nível de serviço (SLA - *Service level agreement*) estabelecidos entre o usuário e o fornecedor da nuvem.

[1] Neste momento, um serviço pode ser a alocação de uma infraestrutura de servidores (*Infrastructure as a Service* – IaaS), uma plataforma para desenvolver aplicações (*Platform as a Service* – PaaS) ou um *software* pronto (*Software as a Service* – SaaS).

[2] Amazon Elastic Compute Cloud (Amazon EC2): http://aws.amazon.com/pt/ec2/

[3] Google Cloud Platform: https://cloud.google.com/

[4] Windows Azure: http://www.windowsazure.com/pt-br/

[5] IBM SmartCloud: http://www.ibm.com/cloud-computing/us/en/

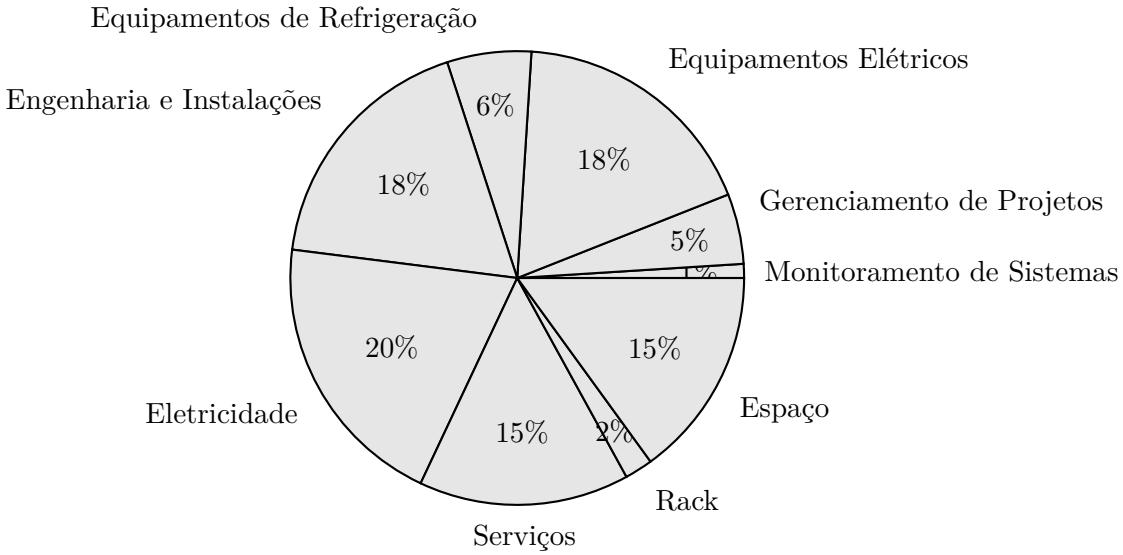


Figura 2.1: *Custo total de posse de um* rack *em um* data center *típico de alta disponibilidade [Ras11]*

2.2 Consumo Energético

Seguindo a tendência comentada na Seção 1.1, os serviços de TI vem apresentando um forte crescimento devido à contínua migração de serviços, análise de dados (*Big Data*, *Business Intelligence*, etc.) e processamentos científicos para grandes data centers. Com isso, o consumo energético total também tem aumentado. Em uma análise do custo total de posse de um *data center* de alta disponibilidade, cada *rack* [6] possui um custo de US$120.000 ao longo de 10 anos [Ras11]. Este custo está dividido conforme apresentado na Figura 2.1.

Como é possível ver na Figura 2.1, os custos relacionados à eletricidade mais os gastos com equipamentos que apenas cumprem o propósito de garantir que o servidor permaneça ligado e refrigerado totalizam 44% (Eletricidade, Equipamentos de Refrigeração e Equipamentos Elétricos). Portanto, uma redução nestes gastos gera um impacto tanto econômico quanto ambiental, com a redução dos recursos naturais necessários para sustentar um *data center*.

2.2.1 Migração de Máquinas Virtuais

Uma das grandes vantagens de manipular máquinas virtuais em um ambiente como o de computação em nuvem é o fato de que é relativamente simples fazer uma realocação de máquinas virtuais de uma máquina hospedeira para outra, mesmo enquanto a máquina virtual está funcionando [VMw] [AU09].

O objetivo desse processo é o de equalizar a infraestrutura utilizada efetivamente com a demanda atual. Em momentos de poucas requisições é possível minimizar o número de nós físicos que estão atendendo a carga de trabalho atual. Enquanto isso, os nós ociosos ficam livres para serem desligados ou colocados em algum estado de hibernação profunda, com baixo consumo energético [BB10]. Quando há um aumento na demanda, é possível realizar o caminho inverso, distribuindo as máquinas virtuais em mais hospedeiros, garantindo o cumprimento do SLA definido entre o provedor da nuvem e o usuário.

2.2.2 Dimensionamento Dinâmico de Tensão e Frequência

A estratégia de dimensionamento dinâmico de tensão e frequência, do inglês *dynamic voltage and frequency scaling*, é uma tática bastante difundida entre os fabricantes de processadores como

[6] O autor define um *rack* como um gabinete vazado de tamanho padronizado (O mais comum é a versão de 19 polegadas de largura e 42 unidades de altura. Cada unidade equivale a 1,75 polegada) e também gabinetes que contém *mainframes* e unidades de *storage*.

uma forma pouco invasiva de economizar energia elétrica. Tecnologias como o Intel Speed Step e o AMD Coll'n'Quiet ajustam automaticamente em hardware a tensão e frequência dos processadores proporcionalmente com suas cargas de trabalho atuais [LMB12].

2.3 Escalonamento de Tarefas

Segundo [LMB12], o escalonamento de tarefas pode ser visto através de duas óticas: a do usuário da nuvem, que deseja que sua tarefa execute o mais rapidamente possível e com o menor custo e a perspectiva do provedor da nuvem, interessado em reduzir os recursos utilizados, gerando economias na manutenção e energia elétrica. Esta monografia foca no escalonamento pelo ponto de vista do provedor.

2.3.1 *Heterogeneous Earliest Finish Time*

O algoritmo *Heterogeneous Earliest Finish Time* – HEFT é um algoritmo de escalonamento de fluxos de trabalho modelados utilizando um digrafo acíclico (*Directed acyclic graph* – DAG) para um número limitado de computadores heterogêneos [KH01].

O HEFT é um exemplo de escalonador *baseado em lista*. Ou seja, funciona ordenando as tarefas a serem executadas ao definir prioridades de escalonamento. Em seguida, processa uma tarefa por vez até que um escalonamento válido seja produzido. Para cada tarefa, é definido o processador que irá acomodar esta tarefa. Todo o processamento do HEFT é feito antes do escalonamento ser executado, ou seja, é um escalonador *off-line*.

Como o problema de escalonamento de tarefas é NP-difícil, o HEFT é frequentemente utilizado como referência na literatura para comparar o desempenho de novas propostas, como por exemplo em [BCdF11].

O HEFT recebe como entrada um conjunto de tarefas modeladas como um DAG, um conjunto de nós computacionais, os tempos para executar uma tarefa em um dado nó e os tempos necessários para comunicar os resultados de uma tarefa para cada uma de suas tarefas filhas no DAG. Como saída o algoritmo gera um escalonamento, mapeando cada tarefa a uma máquina.

O HEFT consiste de duas fases principais:

Fase de priorização Cálculo da prioridade das tarefas e criação da lista de escalonamento com base neste cálculo

Fase de seleção Mapeamento de cada tarefa em uma máquina para processamento

Fase de Priorização

Nesta fase do algoritmo HEFT, cada tarefa deve ser priorizada considerando o comprimento do caminho crítico (Ou seja, o maior caminho) de uma dada tarefa até a tarefa final no fluxo de trabalho. (Passos 1 a 5 no algoritmo Heterogeneous-Earliest-Finish-Time). A lista de tarefas a serem executadas é então ordenada pela ordem decrescente do comprimento do caminho crítico, com empates sendo decididos aleatoriamente. Com essa ordem, é produzida uma ordenação topológica das tarefas, preservando as restrições de precedência do DAG.

A prioridade de uma tarefa n_i é definida recursivamente como:

$$rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)} (\overline{c_{i,j}} + rank_u(n_j))$$

onde n_i representa a i-ésima tarefa, $\overline{w_i}$ é uma média do custo computacional em uma unidade de tempo da tarefa i entre todos os processadores, $succ(n_i)$ é o conjunto de todas as tarefas que dependem imediatamente da tarefa n_i e $\overline{c_{i,j}}$ é o custo de comunicação dos dados transferidos entre as tarefas n_i e n_j na mesma unidade de tempo utilizada antes.

Note que o cálculo de $rank_u(n_i)$ depende do cálculo do $rank_u$ de todas as suas tarefas filhas. A noção intuitiva por trás do $rank_u$ é que ele deve representar a distância esperada de qualquer tarefa até o fim da execução do *workflow*.

Ao processar primeiramente as tarefas que estão em um potencial caminho crítico do DAG, o HEFT possui à sua disposição máquinas mais poderosas disponíveis para o processamento. Desta forma, pode aplicá-las para garantir que este caminho crítico seja processado o mais rápido possível. Um caminho crítico é definido como uma sequência de vértices conectados cujo tempo de execução é o limitante inferior no tempo total de execução do fluxo de trabalho

HETEROGENEOUS-EARLIEST-FINISH-TIME()

1 Defina os custos computacionais das tarefas e os custos de comunicação entre as tarefas com valores médios
2 Calcule $rank_u$ para todas as tarefas varrendo o grafo de "baixo para cima", iniciando pela tarefa final.
3 Ordene as tarefas em uma lista de escalonamento utilizando uma ordem não crescente de valores de $rank_u$.
4 **enquanto** há tarefas não escalonadas na lista
5 Selecione a primeira tarefa, n_i da lista de escalonamento.
6 **para** cada processador p_k no conjunto de processadores ($p_k \in P$)
7 Calcule o tempo mais cedo de conclusão da tarefa n_i, considerando que ela execute em p_k
8 Defina a tarefa n_i para executar no processador p_j que minimiza o tempo mais cedo de conclusão da tarefa n_i.

Fase de Seleção

Na maioria dos algoritmos de escalonamento, o tempo mais cedo de conclusão (*earliest finish time* – EFT) de um dado processador p_j para a execução de uma tarefa é o momento quando p_j completa a execução da última tarefa que foi designada a ele. No entanto, o algoritmo HEFT possui uma política de inserção que considera a possibilidade de inserir uma tarefa em um espaço vago entre duas tarefas já escalonadas em um processador [THW02]. Note que o escalonamento neste intervalo deve obedecer às restrições de precedência.

No algoritmo HEFT, a busca por um *slot* de tempo vago para uma tarefa n_i em um processador p_j começa no momento igual ao tempo que a tarefa n_i estará pronta para executar em p_j, ou seja, o momento quando todos os dados de entrada de n_i foram enviados pelos predecessores imediatos de n_i ao processador p_j. A busca continua até que seja encontrado um intervalo de tempo capaz de suportar o custo computacional de n_i [THW02].

Análise de Complexidade do HEFT

No algoritmo HETEROGENEOUS-EARLIEST-FINISH-TIME, o passo 1 toma tempo $O(e \times p)$ para computar as médias enquanto o passo 2 toma tempo $O(e)$ para computar o comprimento do caminho crítico, onde e é o número de arestas no DAG e p o número de processadores. Para n tarefas a serem escalonadas, o passo 3 necessita de um tempo $O(n \log n)$ para ordenar as tarefas pelo comprimento de seus caminhos críticos. Seja a o número de tarefas que tem n_i como predecessora no DAG, então os passos 5-8 ocupam um tempo $O(a \times p)$ para uma tarefa n_i, assim, o laço enquanto necessita de um tempo $O(e \times p)$. Portanto, a complexidade do algoritmo HEFT é $O(e \times p)$.

Exemplo de Execução

Para uma execução simulada do algoritmo HEFT, consideramos o fluxo de trabalho apresentado no artigo original do algoritmo, [THW02], descrito na Figura 2.2. Cada vértice do DAG é uma tarefa a ser executada em um dos três processadores (heterogêneos) disponíveis: P1, P2 e P3. Os rótulos

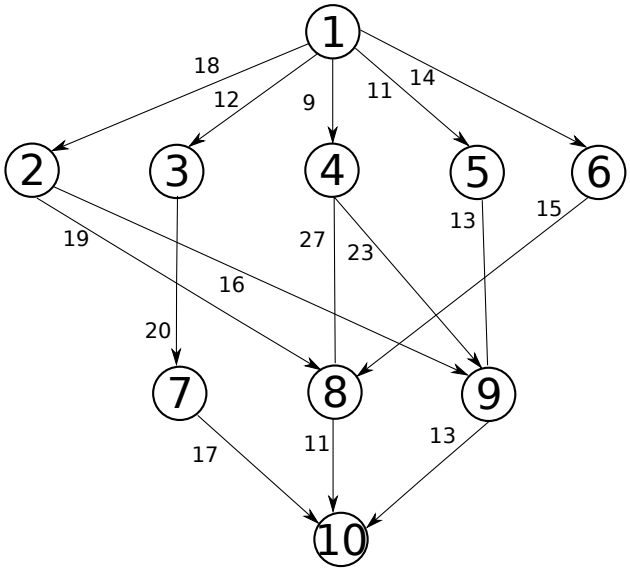


Figura 2.2: *Fluxo de trabalho simulado, contendo os custos médios de transmissão de dados entre um nó de processamento e outro*

nas arestas indicam o tempo necessário para transferir os resultados de uma tarefa entre dois processadores *distintos*.

Para uma dada tarefa, o tempo para processá-la em cada um dos processadores está descrito na Tabela 2.1. Com os valores apresentados é possível calcular o valor de $rank_u$, seguindo a fórmula apresentada anteriormente. O resultado está também na Tabela 2.1. O escalonamento resultante da execução do HEFT com o grafo da Figura 2.2 está apresentado na Figura 2.3. Note que o tempo total de execução do fluxo de trabalho após o escalonamento feito pelo HEFT é de 80 segundos no total. Em comparação, se todas as tarefas fossem processadas na máquina P1 o tempo de execução seria de 127 segundos, na máquina P2 de 130 segundos e na máquina P3 de 143 segundos. Isso confere ao HEFT ganhos de 37%, 38,5% e 44% respectivamente.

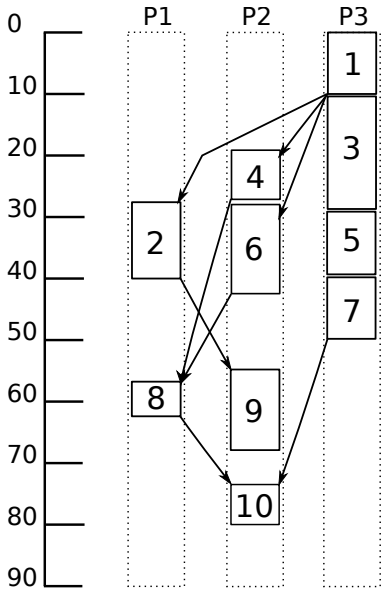


Figura 2.3: *Escalonamento produzido pelo algoritmo HEFT*

Tarefa	P1 (s)	P2 (s)	P3 (s)	$rank_u(n_i)$
1	14	16	9	108.000
2	13	19	18	77.000
3	11	13	19	80.000
4	13	8	17	80.000
5	12	13	10	69.000
6	13	16	9	63.333
7	7	15	11	42.667
8	5	11	14	35.667
9	18	12	20	44.333
10	21	7	16	14.667

Tabela 2.1: *Custos computacionais de uma tarefa em um dado computador*

2.3.2 Embutindo Requisitos de Software no Escalonamento

Notamos neste momento que tarefas a serem executadas em um ambiente de computação em nuvem ou em grade podem requisitar algum software específico para sua execução. Em [BCdF11] é apresentada uma técnica a ser descrita nesta seção para modificar o DAG de dependências entre as tarefas a fim de incorporar tais requisitos de software.

Trivialmente há duas possibilidades para resolver o problema: a primeira possibilidade é alocar apenas uma máquina virtual para cada software distinto a ser executado no fluxo de trabalho. Esta ideia tem o problema de gerar falsas dependências entre as tarefas, já que apenas uma tarefa pode executar por vez em uma máquina e, assim, processos computacionais que poderiam executar em paralelo passam a ter que ser processadas sequencialmente. Outra alternativa é alocar uma máquina virtual para cada tarefa. Esta abordagem além de ser custosa em recursos alocados não necessariamente garante o menor tempo de execução possível pois o tráfego de rede entre os nós de processamento pode não ser o ótimo.

Há uma quantidade exponencial de diferentes formas de alocar máquinas para executar um dado fluxo de trabalho. Esse número vem do fato que alocar tarefas em máquinas é equivalente a encontrar o número de partições de um conjunto, modelado pelo número de Bell, conforme visto nas equações 2.1 e 2.2. O número de Bell é definido como uma recorrência, com B_n sendo o número de partições de um conjunto de tamanho n. Assim, os autores apresentam uma heurística para o problema.

$$B_0 = 1 \tag{2.1}$$

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k \tag{2.2}$$

O objetivo da heurística proposta é reduzir o tráfego de rede necessário e, ao mesmo tempo, evitar aumentar o caminho crítico dos fluxos de trabalho. Isto é feito através da busca das tarefas que possuem os mesmos requisitos de software em um dado caminho do DAG. Dessa forma, há a instanciação de apenas uma VM para cada dependência de software em um caminho.

2.4 Simuladores de computação em nuvem e fluxos de trabalho

Uma vez concebidos novos algoritmos ou abordagens para problemas em computação em nuvem há algumas maneiras de realizar experimentos para estudar o desempenho de tais mudanças: uma possibilidade seria a execução de instâncias reais dos problemas em provedores reais de *cloud computing*. Essa alternativa possui como problema o custo monetário envolvido na instanciação de máquinas virtuais por um tempo prolongado. Ainda, não há o controle sobre o ambiente de

simulação, prejudicando a reproducibilidade dos experimentos. Outra opção seria a de instalar um ambiente próprio para experimentos, novamente esbarrando em problemas financeiros.

Uma alternativa mais factível é a utilização de ambientes de simulação computacional. Estas ferramentas abrem a possibilidade de avaliar uma hipótese em um ambiente totalmente controlado e facilmente reproduzível. Ainda, há um ganho na facilidade de simular diferentes variações de ambientes, facilitando a busca por gargalos na eficiência dos algoritmos utilizados. Esses benefícios vem ao custo da simplificação dos modelos utilizados para simular o ambiente proposto.

Nesta seção serão apresentados os principais simuladores utilizados nos experimentos descritos no Capítulo 4: CloudSim, WorkflowSim e PowerWorkflowSim.

2.4.1 CloudSim

CloudSim [CRB+11] é um simulador para computação em nuvem, reconhecido academicamente com citações em mais de 300 trabalhos indexados pelo Google Scholar [Goo13]. Ele provê as funcionalidades de simulação de máquinas virtuais, *hosts*, *data centers*, políticas de provisionamento de recursos, tarefas a serem executadas em máquinas virtuais (*cloudlets*) além de efetuar análises de duração de simulações e consumo de energia. Ainda, é software livre, disponibilizado sob a licença GNU LGPL [CLO10]. Na Figura 2.4 é possível ver a arquitetura projetada pelos desenvolvedores do CloudSim. Já a Figura 2.5 apresenta o diagrama de classes da implmentação do CloudSim.

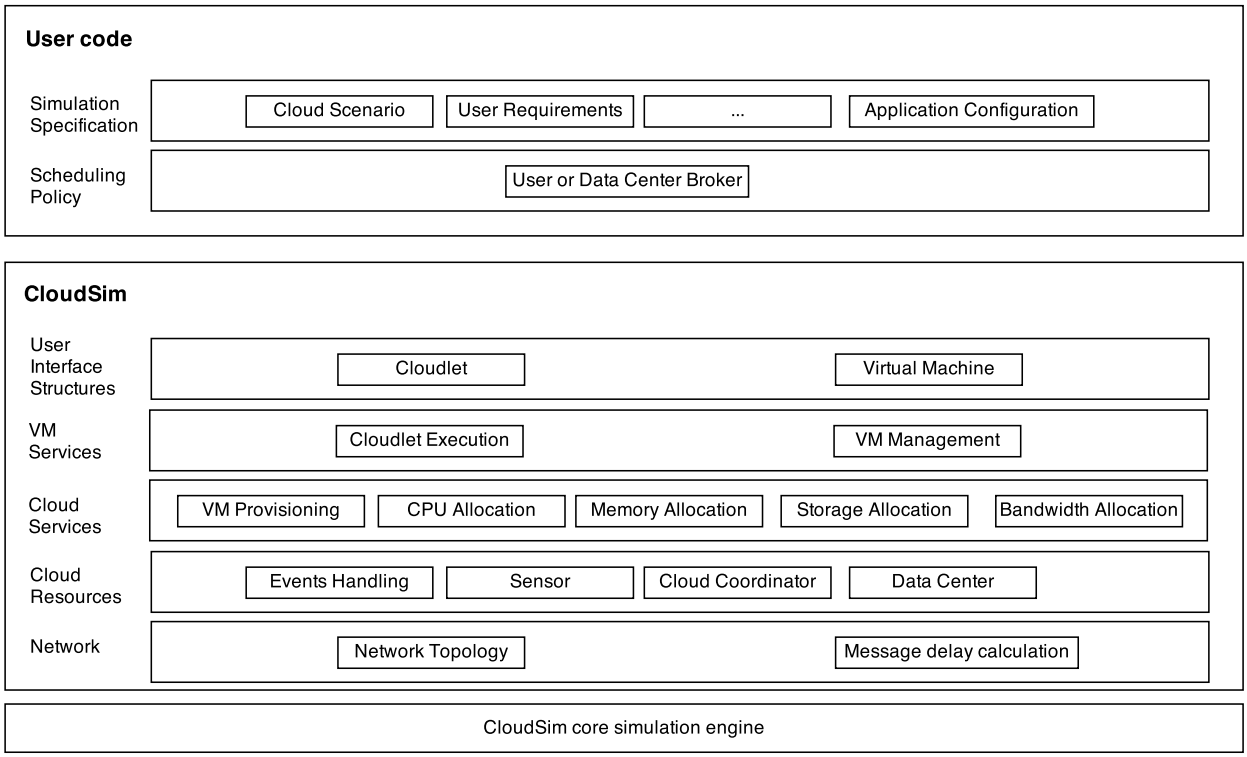


Figura 2.4: *Arquitetura do CloudSim, adaptada de [CRB+11]*

A arquitetura em camadas do CloudSim permite uma separação clara das diferentes possibilidades de extensão e experimentação em um ambiente de computação em nuvem. Por exemplo, todo o código responsável pela simulação energética do CloudSim é na verdade uma especialização das classes originais do simulador. Assim, a classe `PowerDatacenter` e `PowerHost` são classes herdadas de `Datacenter` e `Host` respectivamente.

Com a boa aceitação do CloudSim como uma ferramenta de simulação, passaram a surgir simuladores mais especializados, como por exemplo o WorkflowSim, assunto da Seção 2.4.2.

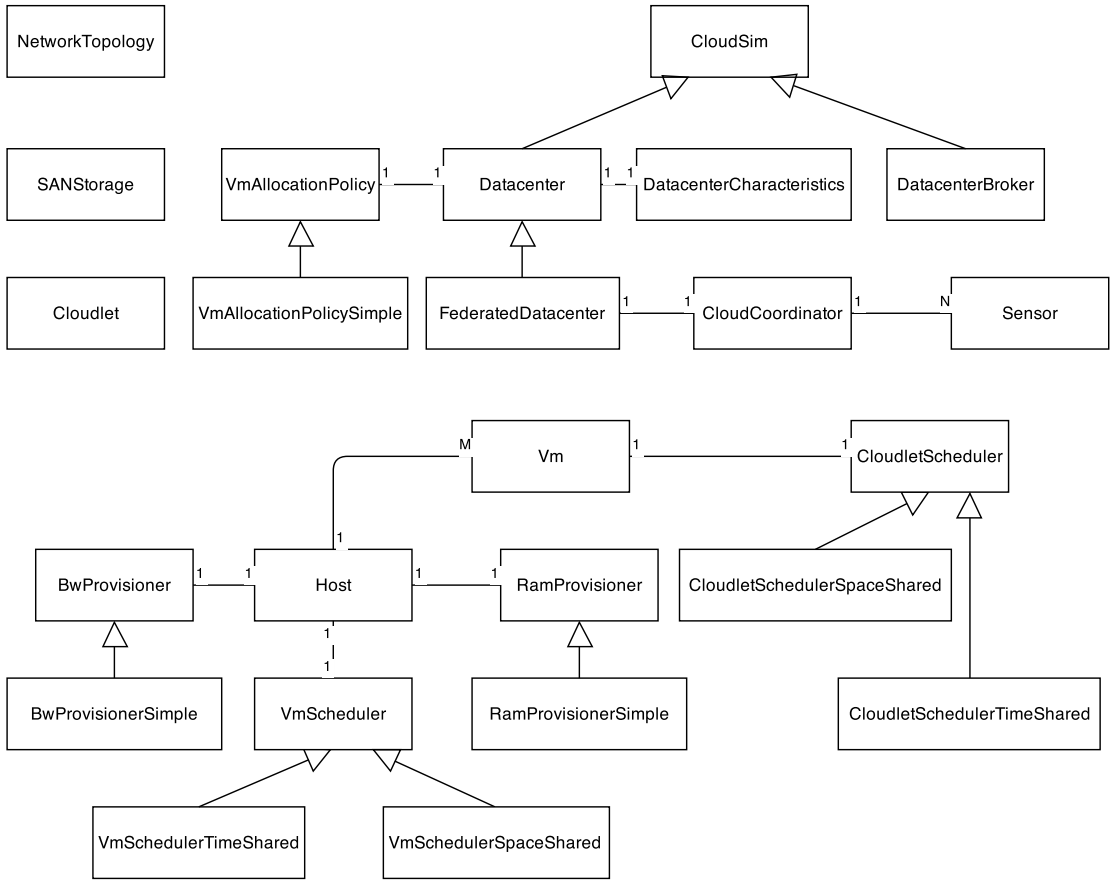


Figura 2.5: *Diagrama de classes simplificado do CloudSim, adaptada de [CRB+11]*

Modelagem de uma Nuvem Computacional

Os serviços de infraestrutura providos por nuvens podem ser simulados através da extensão da classe `Datacenter` do CloudSim. Esta entidade gerencia diversos `Hosts`. Cada `Host` possui uma capacidade de processamento pré determinada, medida em milhões de instruções por segundo (MIPS) e podendo ser com apenas um único ou vários núcleos de processamento, memória e armazenamento. Um `Host` pode incorporar uma ou mais `Vm` (Máquina virtual) de acordo com as políticas de alocação definidas nele pelo administrador da nuvem. Uma política de alocação é definida como as operações relacionadas a uma `Vm` durante seu ciclo de vida: escolha de um `Host`, criação, migração e destruição.

De maneira similar, cada `Vm` possui capacidades de processamento, memória, largura de banda e número de processadores definidos no momento da sua criação. Ainda, pode executar uma ou mais tarefas, `Cloudlets`, obedecendo-se as políticas de provisionamento de aplicações.

Modelagem do Consumo Energético

Como discutido anteriormente, o CloudSim possui uma extensão que incorpora a modelagem do consumo energético das máquinas utilizadas na simulação. Nessa extensão cada elemento de processamento (Tipicamente um núcleo) inclui um objeto que estende o tipo abstrato `PowerModel`, responsável por gerenciar o consumo energético. Isso garante um desacoplamento entre o processamento e a estratégia de modelagem energética empregadas. Por exemplo, um PowerModel pode levar em conta a estratégia de DVFS descrita na Seção 2.2.2 enquanto outro não. O CloudSim apresenta algumas implementações concretas da classe `PowerModel`. A técnica de modelagem empregada é descrita em [BB12].

2.4.2 WorkflowSim

Apesar de ser bem consolidado como ferramenta de simulação de computação em nuvem, o CloudSim não possui certas características necessárias à simulação de *workflows* científicos como por exemplo as ineficiências causadas pelo uso de sistemas heterogêneos e falhas. Ainda, percebe-se a falta de suporte a técnicas de otimização de execução de *workflows* amplamente utilizadas tais como o *clustering* de tarefas. De forma a resolver essas demandas o WorkflowSim foi criado tendo como base o CloudSim [CD12]. A licença do WorkflowSim foi levemente alterada, sua licença é a Globus Toolkit Public License[7] [Che13].

Enquanto o CloudSim se concentra na execução de uma única carga de trabalho, o WorkflowSim foca no escalonamento do *workflow* e sua execução. O último processa *workflows* modelados como DAGs, realizando um escalonamento que obedece a precedência imposta pelo DAG. Ainda, é possível implementar diferentes algoritmos de escalonamento para avaliar suas eficiências. Uma das atividades desse TCC foi implementar o algoritmo HEFT comentado na Seção 2.3.1 no WorkflowSim. [8]

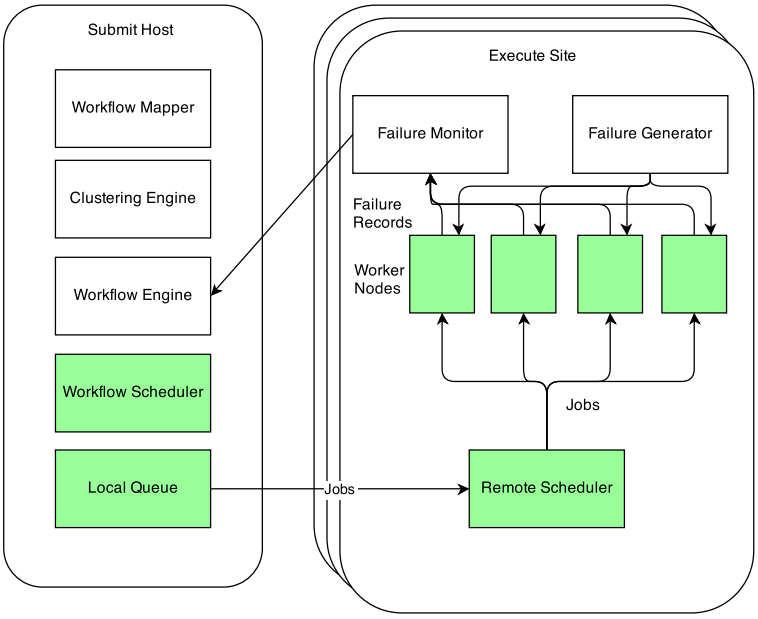


Figura 2.6: *Arquitetura do WorkflowSim, adaptado de [CD12]. Os componentes em verde são providos pelo CloudSim*

Como é possível ver na Figura 2.6, há múltiplas camadas de componentes envolvidos na preparação e execução do *workflow*: Um `Workflow Mapper` que mapeia *workflows* abstratos em *workflows* concretos, dependentes do local de execução; uma `Workflow Engine` que gerencia a dependência de dados e um `Workflow Scheduler` para associar tarefas aos processadores, por fim a `Clustering Engine` é responsável por agrupar tarefas menores em um pacote maior.

2.4.3 CloudSim_DVFS

Um segundo simulador baseado no CloudSim foi desenvolvido de maneira concomitante com o WorkflowSim. O CloudSim_DVFS foi lançado em outubro de 2013 no artigo [GMDC+13]. Esta ferramenta também implementa o processamento de fluxos de trabalho modelados como DAGs e descritos utilizando o mesmo formato de entrada que o WorkflowSim: Arquivos XML no formato DAX definido pelo projeto Pegasus [Peg].

[7] Informações sobre a licença disponíveis em http://www.globus.org/toolkit/download/license.html

[8] Outros algoritmos de escalonamento tais como FCFS, MinMin, MaxMin, MCT e *Data aware* já estão implementados na versão atual (1.0)

Velocidades (GHz)	**0.8**	**1.0**	**1.2**	**1.5**	**1.7**
Nível ocioso (W)	140	146	153	159	167
Nível máximo (W)	228	238	249	260	272

Tabela 2.2: *Frequências do Grid'5000 Reims com as medidas de potência durante cargas mínima e máxima (0% e 100% de uso dos 24 núcleos de um nó de processamento)*

A principal diferença do CloudSim_DVFS para o simulador anterior é o foco em simulações energéticas, como já reflete seu nome. Ele aprofunda a API de simulação energética disponível no CloudSim. Agora, são definidas diferentes estratégias de DVFS:

Performance Utiliza a CPU em frequência máxima (Sem DVFS)

PowerSave Utiliza a CPU na frequência mínima

UserSpace Permite que o usuário defina a faixa de frequência que a máquina deve operar

Conservative Utiliza um limite superior e um inferior para decidir as mudanças na frequência da CPU. Caso a carga esteja acima do limite superior, a frequência é aumentada se possível. Caso ela esteja abaixo do limite inferior tenta diminuir a frequência.

Modelagem energética

Os autores do CloudSim_DVFS optaram pela estratégia "caixa-preta" para a modelagem do consumo energético de uma máquina. Desta forma, é capaz de abstrair detalhes da construção dos processadores atuais e trabalhar em um modelo de alto nível. O modelo descreve dois valores de consumo energético para cada faixa de frequência disponível às estratégias de DVFS: consumo em máxima capacidade (100% da faixa escolhida) e quando a máquina está ociosa (0% da faixa escolhida). Por fim, é feita uma interpolação linear entre estes dois níveis, com base na proporção de tempo que a máquina esteve realizando processamento. A fórmula utilizada está descrita na Equação 2.3. α é o uso de CPU e $P_{CPUIdle}$ e $P_{CPUFull}$ são os valores de potência consumida pela CPU com 0% e 100% de utilização respectivamente.

Para os experimentos propostos no artigo e nesta monografia, foram utilizados os valores experimentais retirados do projeto Grid'5000 REIMS e reproduzidos na Tabela 2.2 [CCD+05] [GMDC+13]. Cada coluna da tabela indica uma diferente configuração de *máquina virtual* e seus respectivos consumos energéticos a 0% e 100% de utilização.

$$P_{TOT} = (1 - \alpha)P_{CPUIdle} + (\alpha)P_{CPUFull} \tag{2.3}$$

Capítulo 3

Algoritmo Proposto

Após a revisão bibliográfica, foi decidido que os algoritmos de referência para comparação com a proposta desenvolvida neste trabalho seriam o algoritmo HEFT, assunto da seção 2.3.1 e os algoritmos apresentados no trabalho de Tom Guérout et al [GMDC+13]. O desenvolvimento de uma proposta de algoritmo de escalonamento de fluxos de trabalho que fosse energeticamente eficiente foi fruto da observação de alguns conceitos importantes:

Uma ideia inicial seria alterar a fase de seleção do algoritmo HEFT para que, ao invés de tentar minimizar o tempo mais cedo de conclusão de cada tarefa processada, tentasse minimizar o consumo energético. Porém, devido à natureza de algoritmo guloso do HEFT, esta ideia foi abandonada.

Para compreender esta decisão é preciso lembrar de um fato. A eficiência energética máxima de um processador (Que pode ser medida em energia consumida por instrução processada) acontece quando este processador opera no máximo de sua capacidade [BH07]. Desta forma, quando o algoritmo HEFT tentar otimizar o consumo energético de cada tarefa individualmente ele optaria por sempre manter a máquina escolhida operando na frequência máxima. De fato, isto é eficiente para um processamento sequencial, mas não necessariamente se mantém verdade para aplicações paralelas. Como maneira de remediar esta possível vulnerabilidade do algoritmo, foi desenvolvida uma versão do algoritmo que emprega uma estratégia de *lookahead*, descrita na seção 3.1.

Paralelamente, foi desenvolvida uma segunda proposta, que faz uso da característica do CloudSim_DVFS que permite que VMs sejam criadas ou removidas durante o momento do escalonamento. Uma variante desenvolvida emprega, ainda, uma estratégia de *clustering* inspirada no trabalho desenvolvido para o WorkflowSim. A descrição detalhada da segunda proposta está na seção 3.2.

3.1 PowerHEFT

A primeira proposta de algoritmo energeticamente eficiente, chamada de PowerHEFT, é uma adaptação do algoritmo HEFT para empregar uma estratégia de *lookahead*. Sua inspiração veio da leitura do artigo [BSM10] que analisa diversas otimizações ao HEFT. Uma das propostas é uma mudança na fase de seleção do algoritmo: ao invés de escolher a máquina que minimize o tempo mais cedo de conclusão de cada tarefa sozinha, seria escolhida a máquina que minimizasse o tempo mais cedo de conclusão das tarefas filhas, após um escalonamento "simulado" com o HEFT. Esta proposta foi chamada de HEFT-Lookahead.

O PowerHEFT funciona da seguinte maneira: como não é sabido de antemão quantas ou quais VMs são necessárias para um processamento que otimize o consumo energético, o algoritmo foi projetado para alocar novas VMs sob demanda. Inicialmente, há apenas uma máquina alocada, a mais rápida disponível. Esta máquina é responsável por receber as primeiras tarefas a serem escalonadas, pertencentes a um possível caminho crítico, como comentado na seção 2.3.1.

Para cada tarefa a ser processada, o algoritmo "força" a alocação de cada tarefa em cada máquina disponível. Em seguida, o algoritmo HEFT é invocado para escalonar todas as tarefas filhas da tarefa atual. Neste momento, o escalonamento parcial possui o seu consumo energético

avaliado, segundo a modelagem energética do trabalho [GMDC+13] e comentada na Seção 2.4.3. A melhor opção de máquina no momento é memorizada.

Em um segundo momento do processamento da tarefa, o algoritmo analisa se seria mais vantajoso energeticamente alocar mais uma máquina para auxiliar o processamento. Neste momento, uma nova máquina é disponibilizada, sendo comparadas as diferentes opções disponíveis para alocação. Em seguida, o algoritmo repete o procedimento descrito no parágrafo anterior.

Uma vez que é conhecida a melhor opção de máquina, o algoritmo aplica o escalonamento para a tarefa atual e prossegue para a próxima tarefa da lista de escalonamento.

É importante ressaltar que algumas tarefas filhas da tarefa atual podem não estar prontas para serem escalonadas por estarem aguardando alguma outra tarefa pai ainda não processada. Nestes casos, a fase *lookahead* ignora a existência destas tarefas pai. Isto tem como consequência o fato que a estimativa energética pode ser mais otimista que a realidade.

O algoritmo do PowerHEFT está descrito abaixo:

```
EscalonarPowerHEFT(tarefa, VM)
1  F = filhos diretos da tarefa no DAG
2  Escalone tarefa em VM
3  Escalone F utilizando o algoritmo HEFT

4  // A modelagem energética utilizada é a descrita na Seção 2.4.3
5  energia = EstimarEnergiaConsumida()
6  retorne energia

Power-HEFT-Lookahead()
 1  // V é o conjunto de VMs usadas ao escalonar
 2  // VmMaisRápida no modelo energético descrito na Tabela 2.2 é a máquina de 1.7 GHz.
 3  V = {VmMaisRápida}
 4  O = os tipos de VMs que podem ser instanciadas
 5  Ordene o conjunto de tarefas segundo o critério rank_u
 6  enquanto há tarefas não escalonadas
 7      t = a tarefa não escalonada de maior rank_u

 8      // Vamos tentar escalonar t em uma VM existente
 9      para cada v em V:
10          EscalonarPowerHEFT(t, v)

11      // Vamos tentar escalonar t em uma nova VM
12      para cada o em O:
13          n = Instanciar(o)
14          V = V ∪ {n}
15          Atualize os valores de rank_u
16          t = a tarefa não escalonada de maior rank_u
17          EscalonarPowerHEFT(t, o)
18          Retorne V e o escalonamento para o começo do laço
19      Escalone t na VM que minimiza a energia consumida
20      Atualize V e rank_u caso necessário
```

3.2 *HEFT Dynamic Allocation of VM*

A segunda proposta de algoritmo baseia-se nas características do simulador CloudSim_DVFS descrito na Seção 2.4.3. Aqui, a motivação para o algoritmo é explorar ao máximo o paralelismo na

execução do fluxo de trabalho. Novamente, o conjunto de máquinas utilizadas é inicializado com apenas uma VM.

Para cada tarefa a ser escalonada é calculado seu tempo mais cedo de *início*. Este instante marca o primeiro momento que a tarefa pode ser executada. Em seguida, o algoritmo HEFT é aplicado para determinar qual será o tempo de início efetivo da tarefa, que depende da ocupação atual das máquinas disponíveis. Sempre que for detectada uma sobrecarga nas máquinas, indicada por tempo de início $>$ tempo mais cedo de início, uma nova VM (A mais lenta disponível, por exemplo a configuração de 0.8 GHz da Tabela 2.2) é alocada. A esperança é que isto garanta uma economia na alocação de VMs sem prejudicar o *makespan* da aplicação.

Uma segunda otimização é a aplicação de *clustering* vertical, o agrupamento de tarefas que podem ser processadas em uma mesma máquina, inspirada no trabalho [CD12]. No parágrafo anterior foi visto que o algoritmo inicialmente aloca apenas máquinas lentas. Isto é interessante caso esta máquina seja pouco utilizada, já que seu consumo energético absoluto é baixo. Por outro lado, esta configuração é ineficiente energeticamente se considerarmos seu consumo energético medido em Joules consumidos por instrução. Assim, quando o algoritmo verifica que seria mais vantajoso alocar mais uma tarefa em uma máquina lenta, esta é "promovida" para ser uma máquina rápida, mais eficiente energeticamente.

O algoritmo do *HEFT Dynamic Allocation of VM* está descrito abaixo:

TaskClustering()

```
1  para cada Vm com tarefas alocadas
2      se VmAnterior e a VmAtual forem do tipo {VmMaisLenta}
3          Aloque uma nova Vm do tipo {VmMaisRápida}
4          Transfira as tarefas da VmAnterior e VmAtual para a nova Vm
```

HEFT-DynamicAllocationVm()

```
 1  Aloque uma Vm do tipo {VmMaisRápida}
 2  Defina os custos computacionais das tarefas e os custos de comunicação entre as tarefas
    com valores médios
 3  Calcule rank_u para todas as tarefas varrendo o grafo de "baixo para cima", iniciando
    pela tarefa final.
 4  Ordene as tarefas em uma lista de escalonamento utilizando uma ordem não
    crescente de valores de rank_u.
 5  enquanto há tarefas não escalonadas na lista
 6      Selecione a primeira tarefa, t_i da lista de escalonamento.
 7      Calcule o tempo mínimo para execução da tarefa t_i com base nas tarefas
        das quais t_i dependa
 8      para cada VM m_k no conjunto de VM (m_k ∈ P)
 9          Calcule o tempo mais cedo de conclusão da tarefa t_i, considerando que ela execute
            em m_k
10      Defina o tempo mais cedo de conclusão da tarefa t_i e o tempo de início da VM
        em que esse tempo foi obtido
11      se a Vm escolhida não é do tipo {VmMaisRápida}
12          Aloque uma nova VM do tipo {VmMaisLenta}
13      senão
14          se a Vm escolhida não é do tipo {VmMaisRápida}
15              Aloque uma nova Vm do tipo {VmMaisRápida}
16              Migre todas as tarefas da Vm antiga
17              Aloque a tarefa na nova Vm
18          senão
19              Defina a tarefa t_i para ser executada na Vm que minimiza o tempo de
                conclusão desta tarefa
```

Capítulo 4

Experimentos

Este capítulo é dividido em quatro seções. Primeiro, na Seção 4.1 é comentada a tentativa de implementação de um simulador, o PowerWorkflowSim para os experimentos realizados nesta monografia. Depois, na Seção 4.2 são apresentados os parâmetros utilizados nas simulações desenvolvidas. Em seguida, na Seção 4.3 são apresentados três fluxos de trabalho que servirão como dados de entrada para os algoritmos de escalonamento. Por fim, na Seção 4.4 são apresentados os resultados experimentais e uma comparação com os algoritmos de referência escolhidos.

4.1 PowerWorkflowSim

Durante a implementação do WorkflowSim os desenvolvedores não tomaram o cuidado de manter a API de simulação energética do CloudSim acessível ao usuário final. De maneira a resolver esse problema, o autor da monografia tentou desenvolver uma versão alternativa do WorkflowSim, o PowerWorkflowSim.

Tanto a API energética do CloudSim quanto o WorkflowSim fazem intenso uso de herança de objetos para reaproveitamento de código e funcionalidades semelhantes. Isto é evidenciado na Figura 4.1, que agrupa um subconjunto das classes principais dos simuladores estudados e suas relações.

Como Java, a linguagem na qual os simuladores foram desenvolvidos, não possui suporte a herança múltipla a estratégia empregada foi a de criar classes herdadas da API energética (`PowerDatacenter` e `PowerVm`, etc) e reimplementar as mudanças propostas pelo WorkflowSim (`DatacenterExtended` e `CondorVm`, etc) nestas novas classes (`PowerDatacenterExtended` e `PowerCondorVm`, etc).

Porém, após três semanas de trabalho lento e minucioso analisando *traces* da execução do simulador não foi possível chegar a um código estável, no qual uma implementação não conflitasse com outra. Com a disponibilização do código fonte do CloudSim_DVFS o projeto de desenvolver o PowerWorkflowSim foi abandonado.

Vale ressaltar que em conversa com o desenvolvedor do CloudSim_DVFS foi possível descobrir que ele também enfrentou os mesmos problemas, tendo que recorrer a caminhos alternativos para desenvolver o seu simulador. As dificuldades enfrentadas por ele foram apresentadas aos desenvolvedores do CloudSim mas estes ainda não identificaram onde aconteceu o problema.

Outro fato relevante a ser notado é que apesar do WorkflowSim não ter sido utilizado como simulador, foram feitas contribuições ao código do simulador. Um *pull request* contendo a implementação do algoritmo HEFT foi aceita para incorporação no repositório do simulador. Ainda, o autor da monografia foi aceito como contribuidor no projeto.

4.2 Ambiente Simulado

O ambiente simulado foi o mesmo que o artigo [GMDC+13] utilizou em seus experimentos e é baseado no Grid'5000 REIMS [CCD+05]. As configurações utilizadas estão na Tabela 4.1.

Configuração	Valor
Número de máquinas físicas (*hosts*)	500
Atraso na inicialização de uma VM	120 ms
Tipos de VMs disponíveis	Descrito na seção 2.4.3
Número de núcleos por *host*	8
RAM por *host*	32 GB
Espaço em disco por *host*	10 TB
Frequência de cada núcleo	1000 MIPS
Latência da rede	0.2 ms
Latência interna	0.05 ms
Largura de banda	1250 Mbps

Tabela 4.1: *Configurações do ambiente simulado*

4.3 Fluxos de Trabalho Utilizados

Para facilitar a análise de algoritmos que operam sobre fluxos de trabalho, tais como os algoritmos descritos nesta monografia, o projeto Pegasus Workflow Management System desenvolveu uma ferramenta geradora de *workflows* científicos. Estes *workflows* artificiais são gerados a partir de informações extraídas de instâncias reais das aplicações juntamente com conhecimentos prévios sobre seu funcionamento interno [Peg12]. Os fluxos gerados pelo projeto Pegasus foram executados no CloudSim_DVFS com as configurações apresentadas na Seção 4.2.

Para este trabalho foram escolhidos três aplicações: Sipht, Montage e CyberShake. As três estão descritas na Figura 4.2. É importante notar a necessidade de realizar testes com fluxos de trabalho com diferentes topologias. Alguns algoritmos de escalonamento podem ser mais adequados a fluxos de trabalho mais ou menos paralelos.

4.4 Resultados Experimentais

Os resultados extraídos das simulações feitas com os três fluxos de trabalho descritos na Seção 4.3 estão descritos abaixo. São feitas análises tanto do consumo energético quanto do tempo de conclusão de cada aplicação (*makespan*). A Figura 4.3 resume esta seção, com os gráficos de uso de energia e tempo para cada um dos *workflows*. Para cada tabela desta seção, valores em negrito indicam o algoritmo que atingiu a melhor performance dentre os avaliados.

4.4.1 Sipht

As Tabelas 4.2 e 4.3 apresentam o *makespan* e o consumo energético respectivamente da aplicação Sipht para todos os algoritmos de escalonamento testados. Neste caso, todos os algoritmos conseguiram superar o modo de referência, Performance para a instância de 30 vértices. No entanto, entradas maiores mostram que o modo OnDemand é mais eficiente para este fluxo de trabalho. A economia energética foi de 15% para a instância com 100 vértices. Já os *makespans* desta instância, exceto o PowerHEFT, estão distribuídos na faixa de [1957 – 2501] segundos.

Ainda, a estratégia de *clustering* (Algoritmos DAVM-TC e Optimal-TC) mostraram-se vantajosos superando a eficiência em ambos os critérios de suas versões regulares.

4.4.2 CyberShake

As Tabelas 4.4 e 4.5 apresentam o *makespan* e o consumo energético respectivamente da aplicação CyberShake para todos os algoritmos de escalonamento testados. Aqui, o comportamento foi similar ao Sipht. Os escalonadores propostos foram eficientes em entradas pequenas mas perderam para o algoritmo OnDemand na comparação com entradas maiores. Novamente o *clustering* se mostrou uma boa otimização em um algoritmo de escalonamento.

Algoritmos	Sipht_30	Sipht_60	Sipht_100
Performance (Sem DVFS)	2349	**1975**	**1957**
[GMDC+13] Optimal	2349	2249	2176
[GMDC+13] OnDemand	2349	**1975**	**1957**
HEFT	2327	2443	2367
PowerHEFT	4016	16330	40239
DAVM	2340	2500	2501
DAVM-TC	2340	2500	2501
Optimal-TC	**2324**	2179	2162

Tabela 4.2: Makespan *em segundos calculado para a aplicação Sipht*

Algoritmos	Sipht_30	Sipht_60	Sipht_100
Performance (Sem DVFS)	3241	5484	9060
[GMDC+13] Optimal	2818	5373	8648
[GMDC+13] OnDemand	**2751**	4669	**7702**
HEFT	3211	6745	10904
PowerHEFT	2925	6006	8886
DAVM	2860	6172	10217
DAVM-TC	2858	6116	10166
Optimal-TC	2784	**5205**	8590

Tabela 4.3: *Consumo energético em Joules calculado para a aplicação Sipht*

Algoritmos	CyberShake_30	CyberShake_50	CyberShake_100
Performance (Sem DVFS)	265	271	**271**
[GMDC+13] Optimal	272	271	**271**
[GMDC+13] OnDemand	265	271	**271**
HEFT	332	**245**	417
PowerHEFT	753	2175	8421
DAVM	263	406	453
DAVM-TC	261	365	436
Optimal-TC	**257**	271	273

Tabela 4.4: Makespan *em segundos calculado para a aplicação CyberShake*

Algoritmos	CyberShake_30	CyberShake_50	CyberShake_100
Performance (Sem DVFS)	380	651	1305
[GMDC+13] Optimal	352	636	1210
[GMDC+13] OnDemand	**323**	**555**	**1114**
HEFT	473	823	1982
PowerHEFT	533	928	1805
DAVM	358	908	2012
DAVM-TC	351	803	1900
Optimal-TC	330	636	1217

Tabela 4.5: *Consumo energético em Joules calculado para a aplicação CyberShake*

Algoritmos	Montage_25	Montage_50	Montage_100
Performance (Sem DVFS)	205	231	236
[GMDC+13] Optimal	205	231	236
[GMDC+13] OnDemand	205	231	236
HEFT	**174**	**179**	188
PowerHEFT	262	419	732
DAVM	180	195	189
DAVM-TC	177	195	**183**
Optimal-TC	205	231	236

Tabela 4.6: Makespan *em segundos calculado para a aplicação Montage*

Algoritmos	Montage_25	Montage_50	Montage_100
Performance (Sem DVFS)	241	544	1111
[GMDC+13] Optimal	241	544	1111
[GMDC+13] OnDemand	204	459	938
HEFT	206	**423**	889
PowerHEFT	307	980	3412
DAVM	200	436	817
DAVM-TC	**195**	438	**784**
Optimal-TC	205	544	1111

Tabela 4.7: *Consumo energético em Joules calculado para a aplicação Montage*

4.4.3 Montage

Já para a aplicação Montage os resultados foram diferentes conforme mostram as Tabelas 4.6 e 4.7. Os escalonadores baseados no HEFT se mostraram melhores que os propostos por [GMDC+13], inclusive superando o modo OnDemand. Neste caso, *clustering* não trouxe ganhos significativos em tempo ou energia.

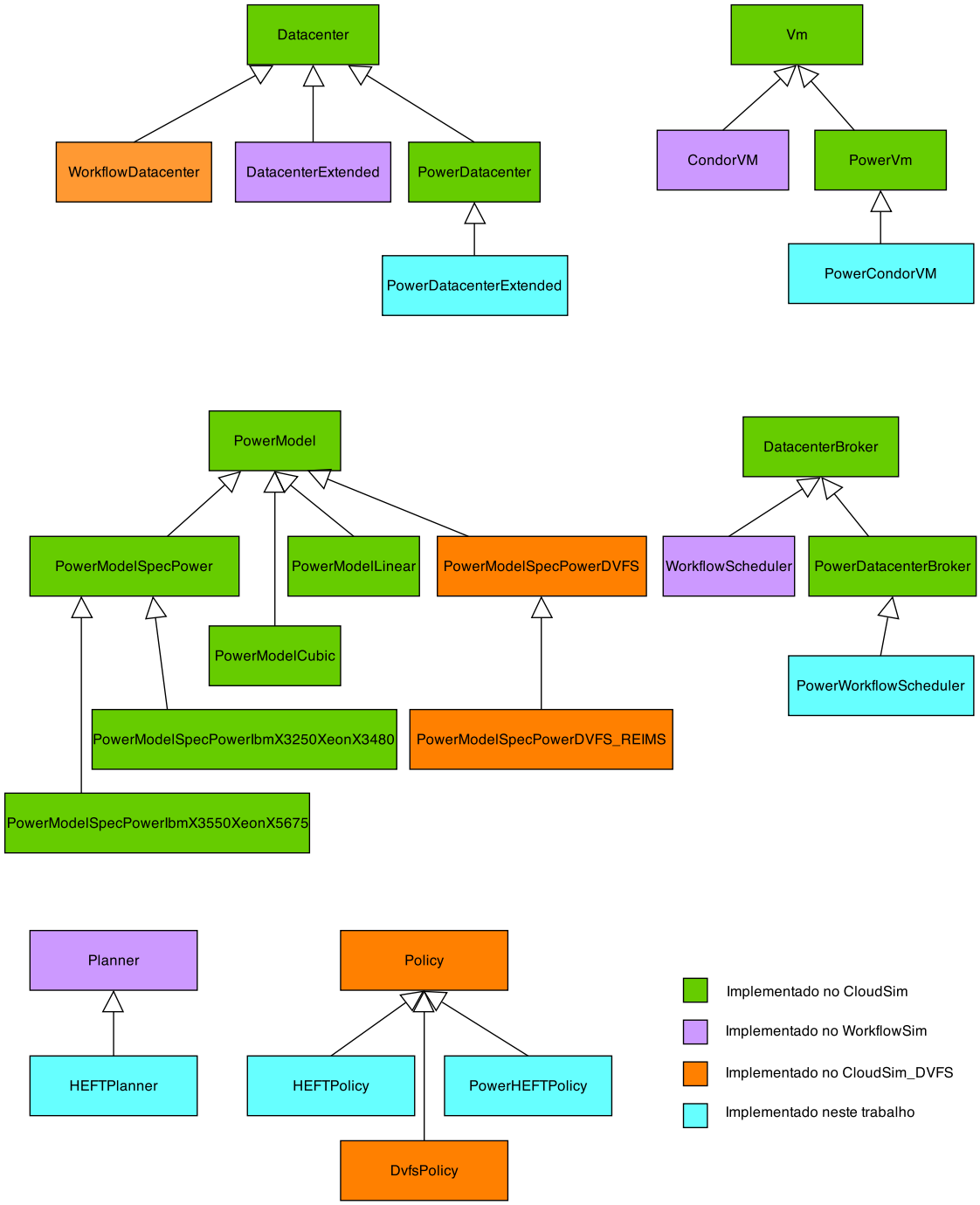
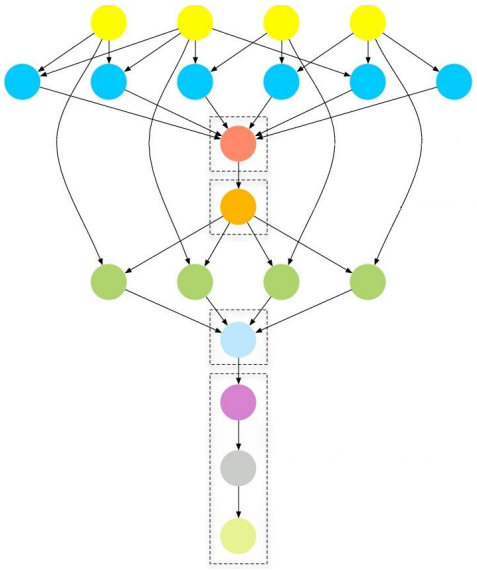
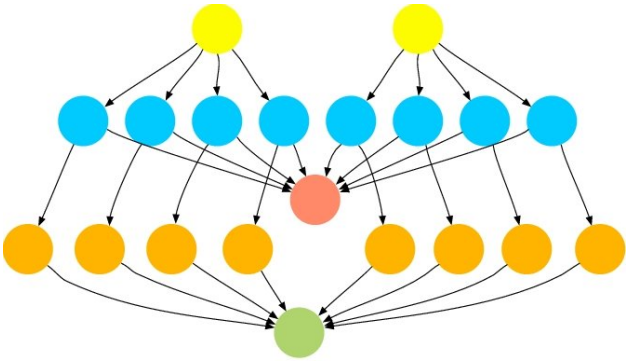


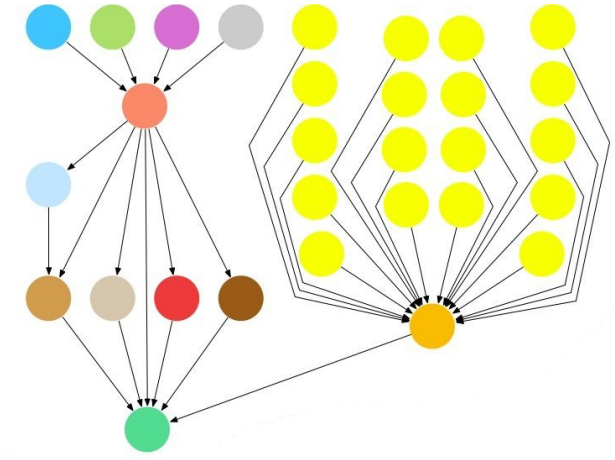
Figura 4.1: *Diagrama de classes parcial dos simuladores utilizados e classes implementadas*



(a) *A aplicação Montage, criada pela NASA/IPAC agrupa diversas imagens de entrada para gerar mosaicos personalizados do céu.*

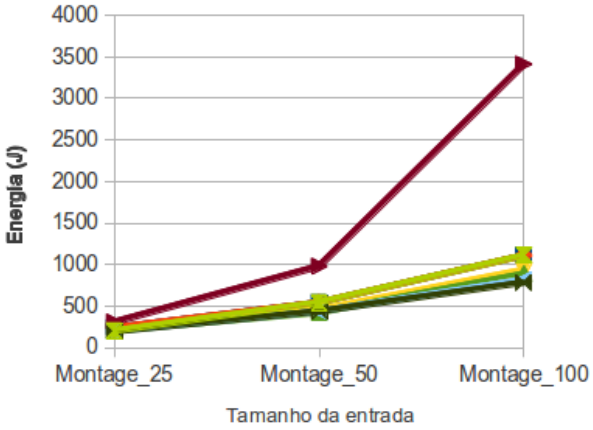


(b) *A aplicação CyberShake é usada pelo Centro de Terremotos do Sul da Califórnia para caracterizar riscos de terremotos em uma região.*

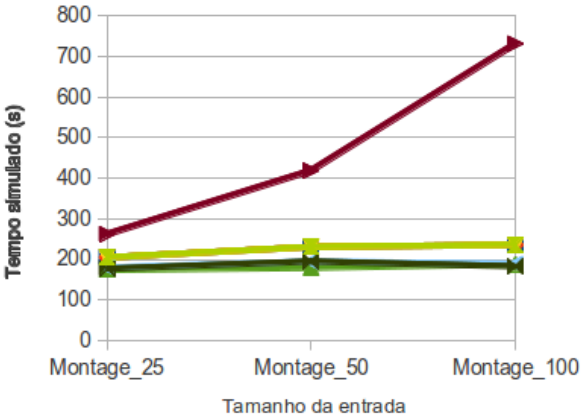


(c) *A aplicação Sipht foi criada pela Universidade de Harvard para automatizar a busca por RNAs não traduzidos (sRNAs) em um banco de dados.*

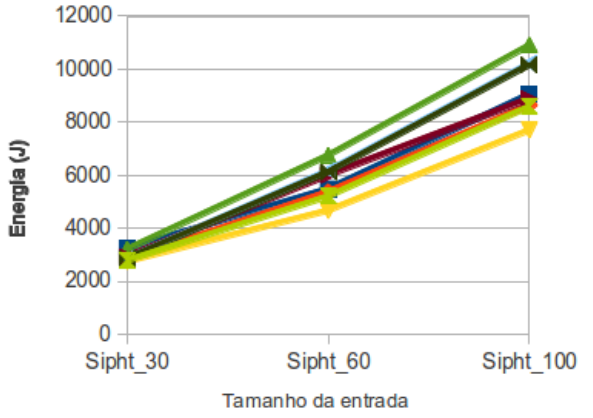
Figura 4.2: *Exemplos de DAGs das aplicações simuladas. Imagens retiradas do Projeto Pegasus, sob a licença Apache.*



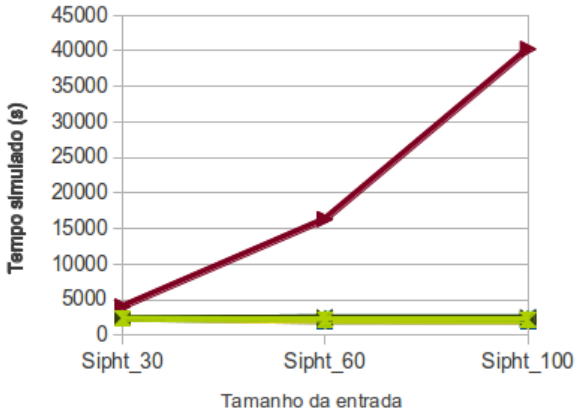
(a) *Consumo energético da aplicação Montage*



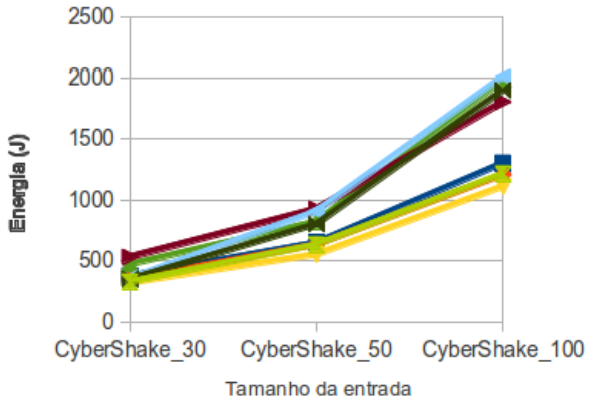
(b) Makespan *da aplicação Montage*



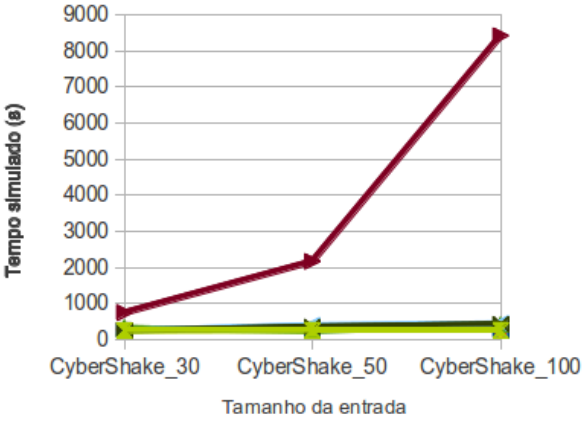
(c) *Consumo energético da aplicação Sipht*



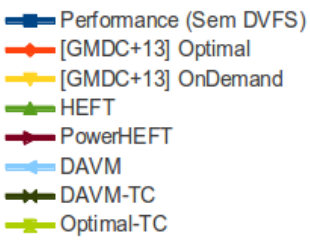
(d) Makespan *da aplicação Sipht*



(e) *Consumo energético da aplicação CyberShake*



(f) Makespan *da aplicação CyberShake*



(g) *Legenda dos gráficos*

Figura 4.3: *Gráficos do consumo energético e* makespan *das aplicações Montage, Sipht e CyberShake.*

Capítulo 5

Conclusões

Nesta monografia apresentada como parte da disciplina Trabalho de Formatura Supervisionado foram estudadas e experimentadas diversas técnicas de escalonamento de tarefas em computação em nuvem. O trabalho contou com a orientação do professor Daniel Macêdo Batista.

No Capítulo 1 foi vista uma motivação para o estudo de técnicas de economia de energia para computação em nuvem. Notou-se um avanço no interesse por uma computação mais econômica no uso de recursos. Ao mesmo tempo, foram definidos os objetivos desta monografia e os seus desafios relacionados, tais como o fato que otimizar o tempo de processamento de uma aplicação paralela é NP-difícil.

O Capítulo 2 pontuou conceitos necessários para a compreensão desta monografia. Foram estudados assuntos como consumo energético e técnicas atuais de economia de energia, um algoritmo clássico da área, o *Heterogeneous Earliest Finish Time* e, por fim, simuladores de computação em nuvem, necessários para os experimentos desenvolvidos posteriormente.

Continuando, no Capítulo 3 foram apresentadas duas propostas de algoritmos de escalonamento com um foco na eficiência energética, o PowerHEFT e o *HEFT Dynamic Allocation of VM*. Em conjunto, foram traçadas as motivações e intuições utilizadas na concepção destes algoritmos.

Por fim, no Capítulo 4 foram apresentados os desenvolvimentos técnicos da monografia, com a tentativa de desenvolvimento de um simulador adaptado para a monografia, as configurações e dados de entrada escolhidos para os experimentos e, por fim, os resultados obtidos e uma discussão.

Esta monografia trouxe algumas contribuições relevantes. Na parte acadêmica, foram desenvolvidos dois novos algoritmos de escalonamento com foco energético. Resultados experimentais mostram que para entradas menores ambos são tão eficientes quanto os outros algoritmos estudados, o que não se concretizou para o caso de entradas maiores.

Já na parte técnica, o autor do trabalho implementou o *HEFT Dynamic Allocation of VM* em dois simuladores de computação em nuvem e disponibilizou o código fonte para os desenvolvedores originais. Por fim, o autor colaborou com diversos pesquisadores internacionais, o que resultou na disponibilização do código fonte de um simulador e na adição do autor como contribuidor de outro simulador.

5.1 Trabalhos Futuros

O algoritmo PowerHEFT mostrou-se pouco escalável no critério de *makespan*. Isto pode ser melhorado de diversas formas, como por exemplo, a definição de um *deadline* para o tempo de execução da aplicação paralela.

A área de escalonamento energeticamente eficiente é muito recente, como é possível ver pelas datas de publicação dos artigos estudados nesta monografia. Uma revisão bibliográfica mais extensa poderia ser feita para encontrar novas propostas de algoritmos que ainda apresentassem margem para melhorias e, em seguida, trabalhar em novas propostas de algoritmos.

Por fim, os simuladores utilizados ainda são muito novos e pouco testados. Foram identificados alguns problemas de engenharia de software, tais como falta de documentação, código morto,

problemas de tradução, etc. que poderiam ser corrigidos.

Parte II

Parte Subjetiva

Capítulo 6

O Trabalho de Conclusão de Curso

Este capítulo é dedicado a comentar alguns dos aspectos vivenciados durante a produção deste trabalho de formatura. Inicialmente, na Seção 6.1 há uma discussão sobre os desafios e frustrações enfrentados e em seguida, na Seção 6.2 são feitas observações sobre a aplicação futura dos resultados alcançados.

6.1 Desafios e frustrações

Um trabalho a ser desenvolvido durante um ano inteiro e primariamente de maneira individual naturalmente enfrenta alguns problemas. É tarefa do autor enfrentá-los e tentar fazer o melhor proveito possível deles. As principais dificuldades estão relacionadas abaixo.

6.1.1 Escopo

Controlar o escopo do trabalho é algo fundamental para garantir que o trabalho seja concluído a tempo e sem grandes sustos. Naturalmente, ao comparar os resultados obtidos com a proposta inicial do TCC pode-se perceber como a proposta é megalomaníaca. Caso a ideia de adaptar o CloudSim para aceitar DAGs como entrada fosse levada adiante, muito provavelmente o TCC não seria sobre um novo algoritmo de escalonamento em computação em nuvem mas sim sobre o trabalho de engenharia de software desenvolvido no ano.

Para gerenciar este problema, pesquisas constantes por material já desenvolvido por outras pessoas foi fundamental para que fosse possível manter o foco no que realmente é o cerne do trabalho. Mas é importante ressaltar que há sempre um compromisso: uma economia de tempo conseguida ao descobrir uma ferramenta que auxilia o seu trabalho é compensado com um aumento de tempo gasto para estudá-la e adaptá-la. Nem sempre o saldo é positivo, como por exemplo com o tempo gasto tentando adaptar o WorkflowSim sem sucesso antes de migrar para o CloudSim_DVFS.

6.1.2 Tempo

É incrível como coisas aparentemente rápidas de serem concluídas podem tomar mais tempo que o planejado. Fazer boas estimativas de tempo para concluir uma tarefa está longe de ser uma ciência exata. Alguns aprendizados conseguidos com o TCC:

Chaveamento de contexto Tentar fazer muitas coisas ao mesmo tempo é uma receita para a ineficiência. O *overhead* presente ao mudar a mente de uma tarefa para outra é uma importante fonte de tempo gasto sem nenhum resultado prático.

Controle do tempo Durante a monografia foi aplicada parcialmente a Técnica Pomodoro [1] os grandes resultados foram controlar o tempo de procrastinação voluntário e involuntário e

[1] Para mais informações, visite http://pomodorotechnique.com/

ainda dividir o tempo de trabalho em parcelas que evitem a fadiga mental adquirida ao manter a concentração por muito tempo em um assunto.

6.2 Observações sobre a aplicação real de conceitos estudados

Este TCC possui um enfoque bastante teórico na área de escalonamento. Os experimentos foram realizados em simuladores, o que garantiu uma economia financeira e maior controle nos resultados. Porém, uma vez que a parte conceitual esteja sólida e os resultados sejam promissores, é possível transportar os trabalhos para ambientes mais realistas. Em última análise, provedores de computação em nuvem são empresas, interessadas por tornar seus processos mais eficientes. E assim, este trabalho possui grande potencial de aplicação prática.

Seja seguindo em uma carreira acadêmica ou na indústria, o aprendizado obtido com os simuladores e os algoritmos trabalhados durante o ano de 2013 não será em vão. Esse pode ser aproveitado em novos experimentos seja na área de escalonamento e energia, seja com outro enfoque.

6.3 Colaboração na produção do trabalho

O desenvolvimento deste TCC teve como ponto forte, a colaboração com pesquisadores internacionais, tai como Weiwei Chen, Rodrigo Neves Calheiros e Tom Guérout. O processo de vencer a timidez para entrar em contato com estas pessoas foi crucial para o avanço dos trabalhos.

Em todos os contatos os pesquisadores foram bastante solícitos, dispostos a colaborar com os trabalhos. Alguns dos resultados foram a minha inclusão como contribuidor do WorkflowSim, implementando o algoritmo HEFT no simulador e a liberação do código fonte do CloudSim_DVFS, que por um lapso dos autores ainda não estava disponível para download.

Questões sociais e profissionais são assuntos ainda muito pouco trabalhados na graduação, apesar de serem pontos vitais para o sucesso profissional de um graduado em Computação. Esta monografia ajudou a suprir um pouco esta lacuna.

Capítulo 7

A Graduação em Ciência da Computação

Realizo neste capítulo um balanço sobre os cinco anos de graduação em Ciência da Computação cursados em duas universidades bastante distintas: a Universidade Federal de Santa Catarina, por dois anos, e a Universidade de São Paulo, por três anos. A primeira graduação, por estar localizada juntamente com os cursos de engenharia da UFSC, tem foco bastante voltado à parte tecnológica da Computação, com grupos fortes fazendo pesquisa em Banco de Dados, Sistemas Operacionais e Engenharia de Software. Já a segunda, localizada dentro do Instituto de Matemática e Estatística, foca muito mais nos estudos de Matemática e conteúdos de fundamentos de Computação como Algoritmos, Grafos e Autômatos.

Felizmente, essas diferenças se tornam complementares quando há interesse do aluno de tentar absorver o que cada lugar tem de melhor. Me considero um sortudo por ter a chance de abraçar essa oportunidade e espero ter aproveitado ao máximo o que me foi oferecido. De fato, encerro esse ciclo com a sensação de dever cumprido, seja em termos acadêmicos, concluindo a graduação com boas notas e cumprindo com meus deveres estudantis, seja em termos sociais, cultivando boas amizades em cada lugar e aproveitando o que a Universidade oferece a seus alunos: restaurante universitário, biblioteca, cursos de idiomas, esportes, viagens acadêmicas, festas universitárias, iniciação científica etc.

7.1 Disciplinas cursadas relevantes para o desenvolvimento do TCC

Desenvolver este TCC foi uma amostra bastante relevante de como os conteúdos em Ciência da Computação estão relacionados, em maior ou menor grau. Felizmente, a área de Redes de Computadores é bastante eclética nas fundações que utiliza para gerar seus resultados. (E que resultados! A Internet é fruto dos esforços de inúmeros pesquisadores em Redes e sempre gerou fascínio em mim. Compreender seu funcionamento e a sua capacidade de mudança foram alguns dos motivos que me fizeram escolher Computação como a carreira que quero perseguir profissionalmente.)

Nesta seção é apresentada uma lista de cursos que fizeram a diferença para o desenvolvimento da minha monografia, direta ou indiretamente, em ordem cronológica.

7.1.1 Programação Orientada a Objetos II

Universidade UFSC

Professor Luiz Fernando Bier Melgarejo

POO II é uma disciplina obrigatória do segundo semestre de Computação na UFSC. O propósito da matéria é cobrir os principais conceitos de Orientação a Objetos e pô-los em prática em um

projeto. O destaque dessa disciplina não é tanto o conteúdo mas sim a metodologia de ensino do professor.

Após uma rápida introdução ao *framework* a ser usado durante a disciplina, os alunos começam desenvolvendo mini-projetos de interesse próprio e usando os (Poucos) conceitos que conhecem de POO aprendidos em POO I. Durante o semestre é obrigação do aluno se oferecer para apresentar os códigos que vem produzindo no(s) projeto(s). É nesse momento que entra o professor, criticando ferozmente o trabalho do aluno, comentando brevemente sobre práticas de programação e padrões de projeto que o aluno não conhecia mas que poderiam ser utilizados. É importante notar que o professor nunca dava a solução do problema. Cabia aos alunos mudar seus códigos por conta própria, pesquisar boas soluções e discutir com colegas para chegar a um consenso do que deve ser apresentado como "solução" em uma aula posterior.

O curioso é que os alunos que sobreviviam a tal provação eram na grande maioria aprovados. Mas mais que apenas uma nota no histórico escolar, os alunos saiam muito mais unidos que antes graças à necessidade de companheirismo para enfrentar o "temido" professor. Além disso, absorviam muito mais conceitos do que numa aula expositiva simples, afinal, ninguém queria ser criticado na frente dos seus colegas então todos pensavam muito em seus códigos antes de fazer a apresentação.

Para este TCC, foi necessário analisar bastante código dos simuladores utilizados além de empregar técnicas vistas nessa disciplina para o desenvolvimento do PowerWorfkflowSim. Foi, também, nessa disciplina que criei a maior intimidade com a linguagem Java, muito utilizada neste trabalho.

7.1.2 Organização de Computadores I

Universidade UFSC

Professor Luiz Cláudio Villar dos Santos

Organização de Computadores é uma disciplina obrigatória do terceiro semestre de Computação na UFSC. O propósito da disciplina é apresentar a interface hardware-software. O destaque novamente vai para o professor. Por um lado, ele é respeitado pelo seu profundo conhecimento de Computação em geral e vivência profissional enquanto que por outro é temido pelo seu alto índice de reprovação.

O professor faz questão de ministrar seu curso tal como fazem as melhores universidades do país. O que pode ser problemático em termos de notas, novamente garante que os alunos que saiam da disciplina estejam preparados para enfrentar problemas com muito mais confiança que em outros lugares.

Para o trabalho de conclusão, foram fundamentais a conceituação de consumo energético e os *tradeoffs* envolvidos na questão velocidade $\times$ economia energética.

7.1.3 Algoritmos em Grafos

Universidade USP

Professor Arnaldo Mandel

Algoritmos em Grafos é uma disciplina obrigatória do quinto semestre de Computação do IME-USP. Aqui, são estudadas as formas de representação computacional de grafos, um pouco de modelagem de problemas usando grafos e diversos algoritmos fundamentais da área.

Grafos e seus algoritmos são uma verdadeira cornucópia de utilidades em Ciência da Computação. Uma estrutura simples e elegante é capaz de modelar e resolver inúmeros problemas e de uma maneira normalmente intuitiva ou palpável. Para um trabalho envolvendo Redes isso não pode ser diferente. Processei DAGs de fluxos de trabalho científicos, apliquei uma busca em profundidade para gerar uma ordenação topológica com o objetivo de executar códigos em diversos nós interligados. Claramente, Grafos foram fundamentais para esse trabalho.

7.1.4 Programação para Redes de Computadores

Universidade USP

Professor Daniel Macêdo Batista

Programação para Redes de Computadores é uma disciplina optativa do curso de Computação do IME-USP. Aqui é feita uma abordagem *top-down* do modelo TCP-IP (Internet) de Redes de Computadores. São estudados os conceitos fundamentais de cada camada, protocolos relevantes e realizados experimentos para reforçar os conceitos aprendidos.

Apesar de já ter cursado Redes de Computadores na UFSC, fazendo um estudo *bottom-up* e estudando redes OSI juntamente com TCP-IP, sentia que meus conhecimentos na área estavam ainda fracos e cursar Programação para Redes de Computadores foi uma ótima decisão. Por ser uma disciplina conjunta com a pós graduação há alunos muito capacitados e interessados no estudo da disciplina. Ainda, o professor consegue em seu curso um feito muito difícil em disciplinas de graduação: motivar os alunos a discutir os assuntos em aula, enriquecendo-a. Por fim, o professor mostra que tem conhecimento prático de programação em redes ao programar e simular os conceitos vistos diretamente na aula.

Para o TCC o professor Daniel aceitou a tarefa de ser meu orientador, muito profissional, trouxe sempre materiais que tornassem o meu trabalho melhor e contribuiu com comentários muito pertinentes.

7.2 Próximos Passos

Com o fim da graduação iminente, surgem as dúvidas de qual carreira seguir. Como comentado no começo do capítulo, considero que aproveitei a universidade nas várias formas possíveis. Na área de pesquisa, participei de duas iniciações científicas diferentes, uma na área de hardware na UFSC e outra na área de ensino de Computação na USP. Assim, considero que este é um bom momento para gerar uma mudança na minha vida profissional, ingressando na indústria de software.

Pretendo continuar trabalhando com Redes e Computação, mas agora em um contexto mais prático. Uma das coisas que mais sinto falta na carreira acadêmica é ver alguma ideia desenvolvida ser utilizada rotineiramente por milhares (Milhões? Bilhões?) de pessoas. Ou seja, ir além do *paper*, da dissertação e da tese. De qualquer forma, mantenho meu carinho especial pelos anos vividos na universidade, de onde levo conhecimento, contatos e amigos para o resto da vida.

Referências Bibliográficas

[AFG+09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica e Matei Zaharia. Above the clouds: A berkeley view of cloud computing, Feb 2009. 5

[AU09] Gustavo P Alkmim e Joaquim Quinteiro Uchôa. Uma solução de baixo custo para a Migração de Máquinas Virtuais. *WPwerformance - VIII Workshop em Desenvolvimento de Sistemas Computacionais e de Comunicação*, páginas 2161–2175, 2009. 5, 6

[BB10] Anton Beloglazov e Rajkumar Buyya. Energy Efficient Allocation of Virtual Machines in Cloud Data Centers. *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, páginas 577–578, 2010. 6

[BB12] Anton Beloglazov e Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. *Concurrency and Computation: Practice and Experience*, 24:1397–1420, 2012. 12

[BCdF11] D.M. Batista, C.G. Chaves e N.L.S. da Fonseca. Embedding software requirements in grid scheduling. Em *Communications (ICC), 2011 IEEE International Conference on*, páginas 1–6, 2011. 4, 7, 10

[BH07] L.A. Barroso e U. Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007. 5, 15

[BSM10] Luiz F. Bittencourt, Rizos Sakellariou e Edmundo R. M. Madeira. Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*, 0:27–34, 2010. 15

[CBdF11] C.G. Chaves, D.M. Batista e N.L.S. da Fonseca. Scheduling grid applications with software requirements. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 9(4):578–585, 2011. 4

[CCD+05] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier e O. Richard. Grid'5000: a large scale and highly reconfigurable grid experimental testbed. Em *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on*, páginas 8 pp.–, 2005. 14, 19

[CD12] Weiwei Chen e E. Deelman. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. Em *E-Science (e-Science), 2012 IEEE 8th International Conference on*, páginas 1–8, 2012. xiii, 13, 17

[Che13] Weiwei Chen. WorkflowSim-1.0 / license-workflowsim.txt, 2013. [Online; acessado em 12 de setembro de 2013]. 13

[CLO10] CLOUDS: The Cloud Computing and Distributed Systems Laboratory. CloudSim: A Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. http://www.cloudbus.org/cloudsim/, 2010. [Online; acessado em 12 de setembro de 2013]. 11

[CRB+11] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose e Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50, Janeiro 2011. xiii, 11, 12

[GMDC+13] Tom Guérout, Thierry Monteil, Georges Da Costa, Rodrigo Neves Calheiros, Rajkumar Buyya e Mihai Alexandru. Energy-aware simulation with dvfs. *Simulation Modelling Practice and Theory*, 39(1):76–91, 2013. 13, 14, 15, 16, 19, 21, 22

[Goo13] Google. Calheiros: Cloudsim: a toolkit for modeling and simulation of ... - google scholar. http://scholar.google.com.br/scholar?cites=272054103506592999&as_sdt=2005&sciodt=0,5&hl=pt-BR, 2013. [Online; acessado em 30 de agosto de 2013]. 11

[KH01] D. Kim e S. Hariri. *Virtual Computing: Concept, Design, and Evaluation*. Advances in Information Security. Kluwer Academic Publishers, 2001. 7

[LMB12] Daniel G Lago, Edmundo R M Madeira e Luiz Fernando Bittencourt. Escalonamento com Prioridade na Alocação Ciente de Energia de Máquinas Virtuais em Nuvens. *XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, páginas 508–521, 2012. 7

[Peg] Pegasus Project. Schema dax-3.4.xsd . http://pegasus.isi.edu/wms/docs/schemas/dax-3.4/dax-3.4.html. [Online; acessado em 19 de novembro de 2013]. 13

[Peg12] Pegasus Project. WorkflowGenerator. https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator, 2012. [Online; acessado em 1 de setembro de 2013]. 20

[PH12] D.A. Patterson e J.L. Hennessy. *Computer Organization and Design: The Hardware/-software Interface*. Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, 2012. 3

[Ras11] Neil Rasmussen. Determining Total Cost of Ownership for Data Center and Network Room Infrastructure. Relatório técnico, Schneider Electric, Paris, 2011. xiii, 6

[Sin07] O. Sinnen. *Task Scheduling for Parallel Systems*. Wiley Series on Parallel and Distributed Computing. Wiley, 2007. 4

[THW02] H. Topcuoglu, S. Hariri e Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260–274, 2002. 8

[VMw] VMware. VMware vMotion: Migração em tempo real de máquina virtual. http://www.vmware.com/br/products/vsphere/features-vmotion. [Online; acessado em 10 de setembro de 2013]. 5, 6