UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO CARIAS DE FRANÇA

MÉTODO GRASP BASEADO EM COMPUTAÇÃO VERDE APLICADO AO ESCALONAMENTO DE RECURSOS E TAREFAS EM GRADES COMPUTACIONAIS

MOSSORÓ - RN
2014

JOÃO CARIAS DE FRANÇA

MÉTODO GRASP BASEADO EM COMPUTAÇÃO VERDE APLICADO AO ESCALONAMENTO DE RECURSOS E TAREFAS EM GRADES COMPUTACIONAIS

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação - associação ampla entre a Universidade do Estado do Rio Grande do Norte e a Universidade Federal Rural do Semi-Árido, para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof.ª Dr.ª Carla Katarina de M. Marques
Co-orientador: Prof. Dr. Carlos Heitor Pereira Liberalino

MOSSORÓ - RN
2014

Catalogação da Publicação na Fonte.
Universidade do Estado do Rio Grande do Norte.

França, João Carias de.

Método GRASP baseado em computação verde aplicado ao escalonamento de recursos e tarefas em grades computacionais. / João Carias de França. – Mossoró, RN, 2014.

90 f.

Orientador(a): Profa. Dra. Carla Katarina de M. Marques

Dissertação (Mestrado em Ciência da Computação). Universidade do Estado do Rio Grande do Norte. Programa de Pós-Graduação em Ciência da Computação.

1. Grades computacionais - Dissertação. 2. Computação verde. 3. Escalonamento. I. Marques, Carla Katarina de M. II. Universidade do Estado do Rio Grande do Norte. III.Título.

UERN/ BC CDD 004

Bibliotecária: Elaine Paiva de Assunção – CRB - 15/492

JOÃO CARIAS DE FRANÇA

MÉTODO GRASP BASEADO EM COMPUTAÇÃO VERDE APLICADO AO ESCALONAMENTO DE RECURSOS E TAREFAS EM GRADES COMPUTACIONAIS

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação para a obtenção do título de Mestre em Ciência da Computação.

APROVADA EM: ___ / ___ / _____.

BANCA EXAMINADORA

_____

Prof.ª Dr.ª Carla Katarina de Monteiro Marques - UERN
Presidente

_____

Prof. Dr. Henrique Jorge Amorim Holanda - UERN
Segundo Membro

_____

Prof. Dr. José Marques Soares - UFC
Terceiro Membro

Ao que tenho de mais importante em minha vida: meus pais, Francisco de Assis e Maria Madalena, meu irmão, Guilherme Carias; e minha prima, Francisca Claúdia.

AGRADECIMENTOS

A minha família, pelo apoio, pelo incentivo, pelos cuidados e principalmente pelo amor incondicional.

A minha orientadora Carla Katarina, pela orientação e apoio que possibilitaram minhas incursões às diversas áreas do conhecimento e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

Ao meu coorientador Heitor Liberalino, por contribuir com grandes ajudas para o caminho da conclusão deste trabalho.

Aos membros da banca de defesa pela coerente avaliação e valiosa contribuição científica nesta dissertação.

Minha gratidão aos colegas de mestrado pelo agradável cotidiano e agonia compartilhadas. Em especial a Jorge Heraldo Danilo.

Agradeço a José Zilton, Luara Almeida e Guilherme Carias por emprestarem os seus computadores pessoais para que fosse utilizados na realização dos testes deste trabalho, permitindo assim, a validação do mesmo.

A UERN e a UFERSA pela oportunidade de aperfeiçoamento acadêmico e infraestrutura fornecida, bem como à CAPES pelo apoio financeiro.

Por fim, a tantos outros que esqueci de colocar aqui. Um grande e fraternal abraço para vocês.

Eu sou o que me cerca. Se eu não preservar o que me cerca, eu não me preservo.

JOSÉ ORTEGA Y GASSET

RESUMO

Motivado por questões ambientais e a demanda por mobilidade e prolongamento do tempo de vida de baterias nos dispositivos moveis, além de garantir um desempenho satisfatório, diversos estudos são abordados para maior eficiência e otimização do consumo energético pelas tecnologias computacionais. Em grades computacionais é possível manter a eficiência computacional e reduzir consumo de energia, através de práticas de Computação Verde no desenvolvimento de algoritmos de escalonamento de recursos e tarefas. Com base em conceitos da Computação Verde, esta dissertação irá propor uma solução baseada na metaheurística GRASP aplicada ao escalonamento de recursos e tarefas em um ambiente de computação em grade. A solução proposta é capaz de reduzir o impacto ambiental devido ao consumo de energia, sem no entanto, comprometer o desempenho dos processos, aproveitando todos os recursos disponíveis por uma grade computacional. É abordado uma grade computacional formada por recursos computacionais físicos, portanto, apresentando resultados na redução do consumo de energia levando em consideração a todos os componentes de um computador físico. O algoritmo implementado apresentou resultados satisfatório, chegando apresentar uma redução do consumo de energia da grade computacional em média de 16% em alguns casos.

Palavras-chave: Grades Computacionais, Computação Verde e Escalonamento.

ABSTRACT

Motivated by environmental issues and the demand for mobility and longer battery life in mobile devices, and ensure satisfactory performance, several studies are discussed for efficiency and optimization of energy consumption by computational technologies. In computational grids is possible to maintain computational efficiency and reduce energy consumption, through practices of Green Computing in the development of scheduling algorithms and resource assignments. Based on concepts of Green Computing, this thesis will propose a metaheuristic based on GRASP applied to the scheduling of resources and tasks in an environment of grid computing solution. The proposed solution is capable of reducing the environmental impact due to energy consumption, without compromising the performance of processes, leveraging all available resources of a computational grid. A computational grid consisting of physical computing resources, thus presenting results in the reduction of energy consumption taking into account all the components of a physical computer is approached. The algorithm implemented with satisfactory results, arriving have reduced the power consumption of the computational grid on average 16% in some cases.

Keywords: Grid. Green Computing and Scheduling.

LISTA DE SIGLAS

ANSI	*American National Standards Institute*
CO_2	Gás Carbônico
ECTC	*Energy Consolidation and Task Consolidation*
EPA	*US Environmental Protection Agency*
ETW	Event Tracing for Windows
IEEE	*Association of Electrical and Electronic*
GIS	*Grid Information System*
GRASP	*Greedy Randomized Adaptive Search Procedure*
HTTP	*Hypertext Transfer Protocol*
MaxUtil	*Maximum Utilization*
MCT	*Minimum Completion Time*
MVs	Máquina Virtuais
RPC	*Remote procedure Call*
REW	Rastreamento de Eventos do Windows
XML	*Extensible Markup Language*
WQ	*Workqueue*
WQR	*Workqueue* com Replicação

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

Diversos estudos estão sendo voltados para maior eficiência energética pelas tecnologias computacionais, principalmente motivados por problemas ambientais, além do objetivo de proporcionar o prolongamento do tempo de vida de baterias e garantir os requisitos de desempenhos sejam atendidos pelos dispositivos móveis (OLIVEIRA; CALVACANTE, 2010).

Segundo Teodoro (2013), arquiteturas paralelas e distribuídas apresentam cada vez mais capacidade de processamento, mas ao mesmo tempo, o ganho de desempenho representa maior consumo de energia.

Já Villarreal (2013) destaca que tradicionamente, os sistemas de computacionais foram desenvolvidos com foco no desempenho e sem a preocupação com a eficiência energética.

Silva C. (2012) afirma que Grades Computacionais tem se mostrado como solução capaz de integrar, em escala global, recursos geograficamente distribuídos e heterogêneos. Entretanto, essa tecnologia possui uma grande demanda de consumo de energia para se manter funcionando, o que tem gerado preocupações não apenas no aspecto financeiro, mas principalmente em relação ao impacto ambiental. Portanto, existe a necessidade de políticas capazes de permitir medidas ecologicamente corretas, sem prejudicar o desempenho computacional.

Tecnologias que utilizam conceitos da Computação Verde visão explorar o máximo dos recursos computacionais minimizando, por exemplo, seu consumo energético (WATANABE et al., 2014).

A computação Verde é uma prática de utilizar recursos computacionais disponíveis de forma otimizar o impacto ambiental. Computação Verde se refere as necessidades da computação usando uma menor quantidade possível de energia ou computação sustentável (KRITHIKA; KEERTHANA, 2013).

Com base em conceitos da Computação Verde, a contribuição deste trabalho descreve uma solução baseada em um algoritmo de escalonamento de recursos e tarefas em um ambiente de computação em grade, capaz de otimizar o consumo de energia, sem no entanto, comprometer o desempenho computacional.

O método em questão é baseado na heurística conhecida como GRASP, proposta por Feo e Resende (1989), com o intuito de permitir o uso mais eficiente do consumo de energia, bem como aproveitar todos os benefícios permitidos pelos recursos de uma grade computacional.

1.1 OBJETIVOS

1.1.1 Objetivos Gerais

O objetivo geral desta proposta de dissertação é propor a implementação de um algoritmo heurístico capaz de ser eficiente no escalonamento de recursos e tarefas no ambiente de grades computacionais, com o objetivo da redução no consumo de energia pelas máquinas que compõe a grade, permitido assim, uma menor degradação ao meio ambiente.

1.1.2 Objetivos Específicos

Buscando uma abordagem a mais ampla possível, ficou definido os seguintes objetivos específicos:

- analisar e registrar os principais fundamentos teóricos da Computação Verde;
- discutir as principais técnicas e estratégias da Computação Verde, principalmente voltadas para a redução do consumo de energia;
- discutir os benefícios proporcionados pela implantação de técnicas da Computação Verde;
- implementar um algoritmo de escalonamento de recursos capaz de proporcionar a eficiência energética em uma grade computacional;
- implementar o algoritmo baseado da heurística GRASP um algoritmo capaz de escalonar recursos e tarefas utilizando técnicas e estratégias da Computação Verde.

1.2 METODOLOGIA

Inicialmente foi feito um levantamento bibliográfico sobre os conceitos e desafios da Computação Verde, estratégias que possam obter maior eficiência no consumo de energia, algoritmos heurísticos.

Depois foi implementado um algoritmo heurístico para escalonamento de recursos em uma grade computacional, onde este algoritmo foi capaz de ser eficiente, além de reduzir o consumo de energia da grade em geral.

Para concluir, foram realizados diversos testes com o objetivo de validar a pesquisa deste trabalho, onde foi montado um grid computacional real com computadores pessoais (notebooks e desktops) e realizado em seguida testes com o algoritmo proposto e com outros algoritmos voltados para escalonamentos.

1.3 ESTRUTURA DO DOCUMENTO

O restante deste trabalho está dividido da seguinte forma:

- O Capítulo 2 apresenta uma visão geral sobre computação verde, sua origem, estratégias verdes e alguns algoritmos de escalonamentos;
- No Capítulo 3 é realizada uma revisão bibliográfica com trabalhos relacionados a este;
- O Capítulo 4 apresenta o modelo energético e a ferramenta Joulemeter usados para estimação de consumo de energia neste trabalho;
- No Capítulo 5 será apresentado o algoritmo proposto;
- No Capitulo 6 será discutido e apresentado os testes e resultados obtidos;
- Ao final, no Capítulo 7, tem-se a conclusão e trabalhos futuros;

CAPÍTULO 2

COMPUTAÇÃO VERDE

Neste Capítulo serão abordados os principais conceitos da Computação Verde, com o objetivo de apresentar uma visão geral, definições, soluções e estratégias verdes, além de um breve histórico da Computação Verde.

2.1 VISÃO GERAL

Existem diversas definições relacionadas ao termo Computaçao Verde, segundo Silva J. (2012), cada uma apresenta além de um conceito, uma abordagem diferenciada de como alcançar as ações sustentáveis.

Murugesan (2008) define a Computação Verde como um paradigma que visa reduzir o impacto ambiental, adotando novas tecnologias e utilizando novas técnicas e materiais, dessa forma, equilibrando a compatibilidade ambiental com a viabilidade econômica e desempenho.

Para Silva J. (2012), ela é encarada sob diferentes perspectivas, que acabam culminando em um único propósito, a redução de impactos ao ambiente natural pelos sistemas computacionais. Murugesan (2008) destaca que a Computação Verde é em parte, o desejo/necessidade da sociedade por tecnologias sustentáveis.

As tecnologias computacionais estão em crescente desenvolvimento, provocando uma ascensão do seu poder de processamento. Entretanto, proporcionalmente ao desenvolvimento dessas tecnologias, tem-se o crescimento do consumo de energia, emissão de Gás Carbônico (CO_2) e lixo eletrônico, entre outros, logo, maior será o impacto ambiental.

Segundo Silva et al. (2010), a necessidade de se tomar medidas ecologicamente corretas se justifica não somente por questões de desempenho, mas por se devolver o equilíbrio ao ecossistema do planeta. Cada vez mais se tem uma atenção para um padrão de consumo de energia para todas as tecnologias de informações e comunicações (ARTHI; HAMEAD, 2013).

Nos últimos anos estudos voltados para maior eficiência no consumo energético em sistemas computacionais tem crescido motivada por questões ambientais.

> A primeira motivação é devida aos problemas causados ao meio ambiente, como as recentes mudanças climáticas, em decorrência do aumento crescente no consumo de energia mundial. A segunda trata-se do desafio principal de prolongar o tempo de vida da bateria dos dispositivos moveis e garantir que os requisitos de desempenho sejam atendidos (OLIVEIRA; CALVACANTE, 2010).

Energia consumida em um sistema computacional compreende, a duas partes (ORGERIE; ASSUNCAO; LEFEVE, 2014):

- A parte fixa (ou estática), que depende do tamanho do sistema e tipo de componente (elementos de computação, armazenamento de dados e de rede); esse consumo é constituído por correntes elétricas presentes em qualquer sistema de potência.
- A parte variável (ou dinâmica) que resulta do uso de computação, armazenamento, e os recursos da rede; causada pela atividade do sistema e mudanças nas taxas de clock.

Segundo Oliveira e Calvacante (2010), a complexidade das aplicações, desde interfaces gráficas mais elaboradas e maiores volumes de dados, geram a necessidade de cada vez mais poder de processamentos, o que consequentemente leva a um maior consumo de energia. Outro motivo para o crescente consumo de energia é a expansão da internet. A digitalização de mídias, a que tem aumentado o número de servidores destinados ao armazenamento. Silva J. (2012) cita o exemplo de uma empresa cujos servidores de *Data Centers* chegam a consumir 2% de toda a energia usada no planeta.

Além do consumo de energia, outro problema que se destaca é a emissão de CO_2, cada fase da vida de um computador (sua produção, tempo de uso e seu descarte), representa de forma direta ou indireta, aumento nas emissões de CO_2 e impacto no meio ambiente (MURUGESAN, 2008).

A Computação Verde (*Green Computer*, em inglês) se apresenta como uma solução eficaz para esse problema. Segundo Zhang, Gong e Li (2011), utilizando tecnologias avançadas de TI, a computação verde pode reduzir emissões e aumentar a eficiência da reciclagem, isto irá proporcionar uma proteção para o meio ambiente e maior conservação de energia.

Além da redução dos problemas citados anteriormente, Villarreal (2013) cita que propociona também a redução de custos operacionais e o prolongamento da vida útil de equipamentos

em ambientes computacionais.

As pesquisas atuais estão voltadas para o uso de computadores com energia eficiente possível, projetar algoritmos e sistemas tecnológicos eficientes (YAMINI, 2012).

Ainda Yamini (2012), existem várias abordagens para a Computação Verde, como: longevidade do produto, algoritmos eficientes, alocação de recursos, virtualização, gerenciamento de energia, etc. Algumas abordagens são descritas posteriormente nesse trabalho.

Em geral, a maior parte das abordagens busca o aumento da eficiência energética de qualquer componente eletrônico. Para tal, diversas técnicas são propostas, tanto em nível de hardware (tais como processador, memória, disco, rede, etc.) quanto em nível de software (funcionalidades presentes no sistema operacional e técnicas que podem ser usadas nas aplicações de usuários) (ALMEIDA, 2012).

A Computação Verde apresenta diversas áreas de focos, podem ser apresentadas como (SILVA et al., 2010):

- Computação com o uso eficiente de energia;
- Gerenciamento de energia;
- Projetos de *Data Centers* Verdes;
- Virtualização de servidores;
- Descarte responsável e reciclagem;
- Utilização de fontes de energia renováveis; e
- Produto de TI com selos ecológicos.

Ainda segundo Silva et al. (2010), a redução do consumo de energia elétrica e de custos, são as principais razões para a utilização de práticas ecologicamente corretas, seguida do menor impacto ambiental e melhoria nos sistemas. O que pode ser visto na pesquisa realizada pela Sun Microsystem australiana (Figura 2.1), com 758 organizações, de grande e pequeno porte, na Austrália e na Nova Zelândia. Onde 25% dos entrevistados afirmam que preferem optar por uma Redução de consumo de energia, além de 18% preferem uma redução de CO_2 e impacto ambiental.

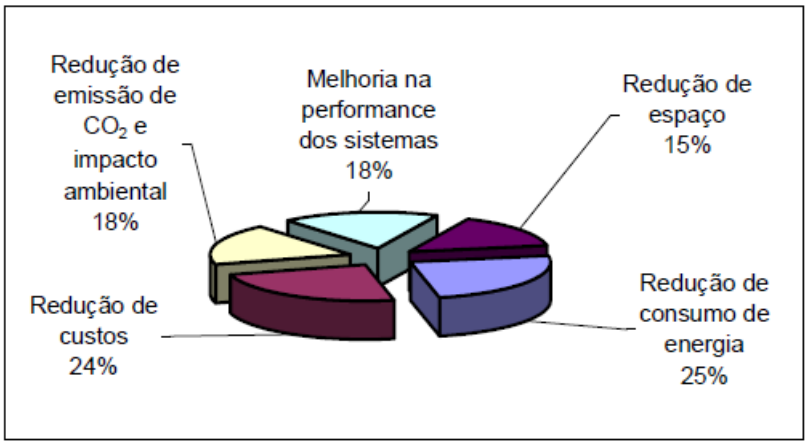

Figura 2.1: Principais Razões para a Utilização de Práticas Verdes. Fonte: (SILVA et al., 2010).

Fabricantes de computadores estão optando por desenvolvimento de "PCs Verdes", utilizando materiais não tóxicos, que consomem menos energia elétrica e são facilmente remontados, permitindo ampliar a sua vida útil (MURUGESAN, 2008).

2.2 ORIGEM DA COMPUTAÇÃO VERDE

A origem da Computação Verde aconteceu no ano de 1987, a Comissão Mundial sobre o Meio Ambiente e Desenvolvimento (*Commission on Environment and Development*) publicou o relatório de título "*Our Commom Future*" (ZHANG; GONG; LI, 2011).

Ainda segundo Zhang, Gong e Li (2011), o relatório propôs conceitos básicos para um "desenvolvimento sustentável", sendo imediatamente aceito por ambientalistas, economistas, ativistas sociais e comunidade internacional.

Entretanto, o inicio do movimento da Computação Verde somente começou a acontecer em 1992, com a criação do plano "*Energy Star*", pela *US Environmental Protection Agency* (EPA), cujo objetivo era criar softwares capazes de serem mais econômicos quanto ao consumo de energia elétrica. O plano foi projetado para reduzir o consumo de energia e diminuir as emissões de gases e efeito estufa (ZHANG; GONG; LI, 2011).

> No ano de 2006, sobre a operação *United States Green Electronics Council*, a *American National Standards Institute* (ANSI) formulou a norma de número 1680 da *Association of Electrical and Electronic* (IEEE), e implementou o sistema de certificação EPEAT, como uso para certificar produtos eletrônicos sobre o impacto

ambiental durante o seu ciclo de vida (ZHANG; GONG; LI, 2011):.

Atualmente, a Computação Verde é considerada umas das tecnologias mais promissoras para o momento da indústria da computação, além de ter ganho popularidade em empresas de TI, institutos de pesquisas e de governos (LI; ZHOU, 2011).

Um exemplo é a produção de processadores, como citado por Almeida (2012). Os processadores estão ficando cada vez mais "verdes", já que as novas gerações desses componentes apresentam transistores de dimensões reduzidas, requerendo menos energia para funcionar.

2.3 ESTRATÉGIAS VERDES

Diversas áreas são dedicadas com o objetivo de alcançar uma computação sustentável. Pode-se citar estratégias como Projeto de *Data Center verde*, Virtualização, *Cloud Computing* (Computação em Nuvem) e *Grid Computing* (Grade Computacional). Tais estratégias são abordadas a seguir:

2.3.1 Data Centers verde

Data Centers são instalações que contêm equipamentos eletrônicos para uso de processamento, armazenamento de dados e comunicação de redes, comuns e essenciais em empresa, centros acadêmicos e sistemas de governos (WANDERS, 2011).

Os *Data Centers* são grande geradores de aspectos negativos e oferecem altos riscos para o meio ambiente. Segundo Silva J. (2012), tais aspectos são: alto grau de emissão de CO_2, alto consumo de energia, custos altos, crescimento não planejado, entre outros fatores.

> A maioria dos *Data Centers* existentes não foram construídos para suportar a densidade de calor e consumo energéticos típicos dos servidores e soluções de armazenamento atuais. O gerenciamento da capacidade de energia, refrigeração e contingência está em crescimento de importância nas organizações pelas restrições de espaço físico e devido ao nível de crescimento da temperatura média dos equipamentos (WANDERS, 2011).

Logo, com o objetivo de reduzir o impacto ambiental por essa tecnologia, surgiu a iniciativa do projeto de *Data Centers* Verdes. Essa iniciativa consiste na ideia de recuperar a capaci-

dade de energia e resfriamento de um *Data Center*, reduzindo ao mesmo tempo, os custos e o consumo de energia (SILVA J., 2012).

Ainda segundo Silva J. (2012), a criação de *Data Center Verde* pode ser uma tarefa complexa, entretanto, existe diversas soluções e técnicas disponíveis. A Tabela 2.1 apresenta algumas dessas técnicas para reduzir o consumo de energia em um *Data Center*.

Tabela 2.1: Técnicas de como reduzir o consumo de energia em *Data Certers*. Fonte: (SILVA J., 2012).

O que fazer?	**Economia potencial**	**Plano de Ação**
Dimensão correta da Infraestrutura física	10% a 30%	Desenha uma arquiteruta modular, com fornecimento de energia escalável e sistemas de refrigeraçao. A economia é maior em ambiente redundates.
Virtualização de servidores	10% a 40%	Instalar software de virtualização e consolidar aplicativos em menos quantidade de servidores.
Sistema de refrigeração eficiente	7% a 15%	Criar rotas curtas de ar, o que requer menos ventilação.
Layout de piso	5% a 12%	Dispor as máquinas em corredores de ar quente e frio, para que o calor de uma não prejudique a ventilação da outra.

2.3.2 Virtualização

A virtualização pode ser definida como sendo uma tecnologia capaz de dividir um computador em diversas máquinas independentes que podem suportar diferentes sistemas operacionais e aplicativos executados concorrentemente (RUEST; RUEST, 2009 apud SILVA; SANTOS, 2010).

A virtualização se apresenta como uma das formas mais eficientes para se reduzir custos de energia, aquisição de novos equipamentos, reduzir espaço físico utilizado por equipamentos computacionais, além da emissão de gases nocivos (SILVA; CARVALHO, 2011). Segundo Arthi e Hamead (2013) esta tecnologia tem sido amplamente utilizada em centros de dados moderno para melhorar a sua eficiência energética.

Um dos principais pontos a favor da virtualização é a redução de componentes eletrônicos, pois causam um enorme impacto ao meio ambiente, através da diminuição de equipamentos de hardware (SILVA; SANTOS, 2010).

Silva e Carvalho (2011) citam que a TI da Microsoft estimou que, em média, uma máquina virtual com a utilização típica requer energia em cerca de 90% menos do que um servidor físico.

Experiências apontam que apenas um servidor virtual pode melhorar em 60% seu desempenho e que a eliminação desse servidor reduz de aproximadamente 200 a 400 W (*watts*), dependendo da tecnologia adotada (SILVA et al., 2010).

2.3.3 Computação em Nuvem

A Computação em Nuvem através de uma conexão com a internet, permite acessar uma grande diversidades de recursos tecnológicos espalhados em diversos servidores remotos.

A Computação em Nuvem pode ser definida como Rocha (2013):

> Um modelo para a oferta de serviços sob demanda na Internet. Neste, um provedor de serviço de nuvem oferece serviços de processamento e armazenamento de informação por meio da virtualização de uma infraestrutura computacional, composta de servidores, equipamentos de comunicação, sistemas de armazenamento de dados, aplicativos, dentre outros.

A Computação em Nuvem é uma tecnologia que proporciona o armazenamento, gestão e disponibilidade de dados, software, aplicações e/ou serviços computacionais (FERNáNDEZ; MARCELINO; MARQUES, 2011).

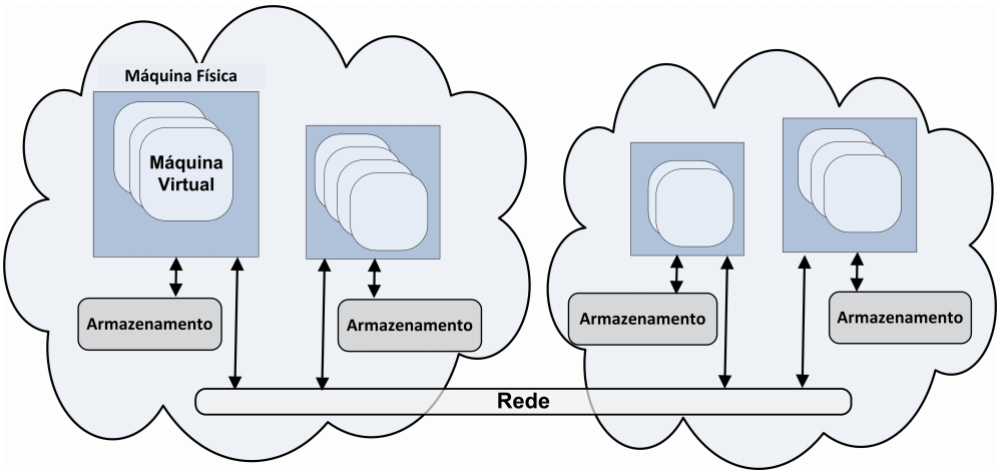

Figura 2.2: Ambiente de computação em nuvem (SOUSA et al., 2011).

A Computação em Nuvem segue o caminho da participação colaborativa, utilizando a memória e as capacidades de cálculo e armazenamento de computadores e servidores partilhados

e ligados através da internet (SILVA A., 2012).

> A Computação em Nuvem pode ser definida como: paradigma de computação distribuída que é impulsionada por economias escalonáveis, em que um grupo abstrato, virtualizado e dinamicamente escalável, gerencia o poder computacional de armazenamento e plataformas, e serviços são entregues sob demanda para clientes externos através da Internet (FOSTER et al., 2008 apud STANOEVSKA-SLABEVA; WOZNIAK; RISTOL, 2010).

A infraestrutura do ambiente de computação em nuvem normalmente é composta por um grande número, centenas ou milhares de máquinas físicas ou nós físicos de baixo custo, conectados por meio de uma rede (SOUSA et al., 2011) como ilustra a Figura 2.2. Cada máquina física tem as mesmas configurações de software, mas pode ter variação na capacidade de hardware em tempos de CPU, memória e armazenamento de disco (SOROR et al., 2010). Dentro de cada máquina física existe um número variável de máquinas virtuais (VM) ou nós virtuais em execução, de acordo com a capacidade de hardware disponível na máquina física (SOUSA et al., 2011).

Essa tecnologia permite a realização de atividades simples, tais como citado por Silva J. (2012): deixar um documento na rede, usar aplicativos que não estão instalados no computador em uso, usar uma conta de e-mail, etc. São exemplos de atividades rotineiras.

Essa solução computacional se justifica por apresentar uma redução de custos, de consumo de energia, trabalho humano, manutenção, entre outras. Tal medida é capaz de fornecer menores cursto, além de redução no impacto ambiental gerado pelo setor de tecnologia (SILVA J., 2012).

2.3.4 Grade Computacional

Grades Computacionais podem ser definidas como sendo uma infraestrutura de hardware e software capaz de fornecer acesso seguro, consistente, abrangente, de baixo custo e capaz de apresentar alta qualidade de capacidade computacional (FOSTER, 2002).

Já Soares e Faina (2008), define como sendo um sistema de suporte à execução de aplicações paralelas que acoplam recursos heterogêneos distribuídos, oferecendo um acesso consistente e barato aos recursos, independente de sua posição geográfica.

O termo Grade Computacional (*computational grid*) é uma analogia com a rede de energia elétrica (*electric grid*)(SANTOS NETO, 2014), por fornece energia elétrica de forma transparente e sob demanda. Ainda Santos Neto (2014), a proposta de um grade computacional é ser uma infraestrutura de computação que atenda às necessidades computacionais dos usuários abstraindo os detalhes de como esse atendimento é realizado.

> Uma grade computacionais deve prover acesso consistente aos recursos disponíveis, fazendo com que as descontinuidades físicas (diferenças entre tipos de hardware, sistemas operacionais e canais de comunicaçao) se tornem completamente transparentes aos usuários (BORRO, 2014).

A Computação em Grade, permite que seja possível alocar grandes quantidade de recursos em uma aplicação paralela, reduzindo custos em relação a alternativas tecnológicas tradicionais.

Esse paradigma permite agregar recursos de alto desempenho computacional e formação de Organizações Virtuais (SILVA J., 2012).

Tal capacidade vem sendo desenvolvida nos últimos anos por causa de alguns fatores como: o aumento de desempenho e a redução do custo dos microprocessadores e das redes de forma em geral (SILVA J., 2012), .

As Grades Computacionais permitem que recursos possam ser distribuídos pelo mundo e utilizados de forma dinâmica e intercambiável com base no clima e condições ambientais, para minimizar o consumo de energia e custos associados (STANOEVSKA-SLABEVA; WOZNIAK; RISTOL, 2010).

2.4 COMPUTAÇÃO VERDE A PARTIR DE COMPONENTES DE SOFTWARE

O estudo com foco na redução de energia de sistemas computacionais e eletrônicos, tem sido voltado, frequentemente, aos componentes de hardware Segundo (SILVA J., 2012).

Além das estratégias anteriormente descritas na Seção 2.3, a Computação Verde tem abordado desafios quanto à natureza algorítmica. Problemas algorítmicos envolvem diretamente questões de gestão de poder, energia ou temperatura com um recurso (PRUHS, 2011).

A problemática algorítmica surge porque novas tecnologias apresentam a necessidade de redução de consumo de energia, sem que afete o seu poder computacional.

Segundo Li e Zhou (2011), pode-se atribuir a responsabilidade do consumo de energia a aplicativos individuais, que podem também contribuir com a redução do calor produzido e a eletricidade consumida.

Algumas estratégias da Computação Verde a partir de componentes de software são abortadas a seguir:

2.4.1 Gerenciamento de Hardware

Outra forma de economia de energia através do gerenciamento de hardware, quanto ao uso de um determinado dispositivo de hardware. Aplicações devem informar ao sistema operacional que não irá mais utilizar o dispositivo (ALMEIDA, 2012). Se não existir a informação do que não será utilizado, mesmo com a aplicação terminada, o sistema operacional vai manter o dispositivo em estado ativo, desperdiçando energia.

A ideia do gerenciamento de hardware consiste basicamente em informar ao sistema operacional que um determinado hardware não será mais utilizado, logo, desligando este dispositivo. Evitando assim, o consumo de energia.

2.4.2 Algoritmos eficientes

O estudo de Algoritmos eficientes tem o objetivo de projetar algoritmos que sejam capazes de reduzir o tempo de execução e demanda por recursos. Algoritmos eficientes proporcionam uma redução de custo/consumo dos recursos computacionais.

Um dispositivo energeticamente eficiente consome energia de forma proporcional à quantidade de trabalho que o mesmo produz (GUNARATNE; CHRISTENSEN; NORDMAN, 2005 apud ALMEIDA, 2012).

Um dos benefícios de algoritmos eficientes na prática é com relação a máquinas ociosas, pois consome menos energia, logo, quanto mais rápido possível terminar a execução e o algoritmo encontrar a solução do problema, mais tempo a máquina irá passar no estado ocioso.

Programadores podem implementar aplicação com paralelismo de dados e paralelismo de *thread*, já que permitem a redução no tempo de execução do algoritmo. Segundo Oliveira e Calvacante (2010), esses paralelismos permitem benefícios de desempenho e redução do consumo de energia.

Entretanto, nem sempre esse ganho é possível devido geralmente a problemas de sincronização no uso do paralelismo (OLIVEIRA; CALVACANTE, 2010).

Um algoritmo pode ser analisado de duas formas: quanto a corretude e quanto a complexidade.

A análise da corretude vise determinar se o o algoritmo chega na solução desejada (COSTA et al., 2014). A outra a é desitanada ao tempo de execução, como descrito por Cormen et al. (2002), em ocasiões que recursos de memória, largura de banda de comunicação ou hardware, são as principais preocupações, mas com frequência estamos preocupados em medir o tempo de computação. A análise de complexidade determina qual o custo de um algoritmo na busca de uma solução, principalmente o custo em termos de tempo de execução (COSTA et al., 2014).

O tempo de execução de um algoritmo é representado pela função de custo *T(n)*, que representa a média de tempo necessário para executar um algoritmo para um problema de tamanho n (SILVA J., 2012). Segundo Cormen et al. (2002), *T(n)* não representa diretamente o tempo de execução, mas o número de vezes que as operações relevantes são executadas.

De acordo com Ascencio e Araujo, citado por Silva J. (2012), valores pequenos de *n* (entrada), em qualquer algoritmo, mesmo que ineficiente, gastará pouco tempo. Entretanto, o que leva a escolha de um algoritmo é o desempenho do mesmo sobre tamanhos de entrada grandes.

2.5 ALGORITMOS DE ESCALONAMENTO

Nesta seção serão abordados alguns algoritmos utilizados para escalonamento de recursos em grade computacionais.

2.5.1 Sufferage

Segundo Guadagnin (2010):

> O algortimo *Sufferage* é uma heurística em que cada tarefa deveria ser realizada na máquina onde houvesse menor sofrimento (no caso, menor tempo de execução). O *Sufferage* é o valor da diferença entre o melhor *Minimum Completion Time* (MCT)[1] e o segundo melhor MCT de cada tarefa. Ou seja, quando cada ta-

[1] O MCT é uma previsão do tempo de conclusão de uma tarefa em um determinado recurso. A forma como é calculado o MCT depende de cada implementação dos algoritmos.

refa seria prejudicada se não fosse executada na máquina com a maior capacidade de processamento (GUADAGNIN, 2010).

O Sufferage é um algoritmo dinâmico que tem mostrado atingir bom desempenho em ambientes heterogêneos e dinâmicos quando considerado a existência de informação completa (OLIVEIRA et al., 2010).

2.5.2 XSufferage

O algoritmo *XSufferage* é uma heurística de escalonamento que se baseia em informações sobre o desempenho dos recursos e da rede que os interliga (CASANOVA et al., 2010).

É uma heurista baseada no *Sufferage*, onde a principal diferença entre o *Sufferage* e o *XSufferage* é a forma de como é calculada o menor tempo de execução. Segundo Guadagnin (2010), o *Xsufferage* não se baseia apenas no menor tempo de execução, mas também leva em consideração a disponibilidade da CPU, da rede e os tempos de execução das tarefas.

O escalonador observa sempre os recursos livres no momento de escalonar a tarefa, caso contrário sempre o recurso mais rápido da rede receberia tarefas (FALAVINHA JUNIOR et al., 2007).

2.5.3 Max-Min

O *Max-Min* é uma heurística que tem como objetivo alocar tarefas de maior carga para recursos que possuam maior capacidade de processamento.

Incialmente, nenhuma tarefa se encontra mapeada, então, o algoritmo realiza uma busca procurando o conjunto de menor MCT de cada tarefa, ou seja, em qual máquina determinada tarefa pode ser executada em menor tempo (GUADAGNIN, 2010). Deste conjunto, é selecionado o maior MCT ou a tarefa com a maior carga. Finalmente, as tarefas vão sendo mapeadas uma a uma até que todas sejam executadas (GUADAGNIN, 2010).

2.5.4 Min-Min

O *Min-min* é uma heurística que se basea no MCT. Esse algoritmo recebe como entrada um conjunto de tarefas não mapeadas M e um conjunto de máquinas constantes da grade (GUADAGNIN, 2010).

Ainda Guadagnin (2010), ao iniciar, o algoritmo calcula o tempo de conclusão que cada tarefa gastaria para ser concluída em cada máquina. Em seguida, o algoritmo irá buscar o menor tempo de execução da tarefa e sua respectiva máquina para execução.

Por fim, o *Min-min* busca a tarefa com menor tempo de conclusão dentre todas as tarefas de *M* e realiza a atribuição da tarefa para execução. Depois de mapeada, a tarefa é removida de *M* e seu tempo de execução é incrementado na respectiva máquina afim de controlar seu tempo de ocupação (GUADAGNIN, 2010). Enquanto existir tarefas, o processo descrito é repetido.

2.5.5 Workqueue

Workqueue (WQ) é um algoritmo de escalonamento que não usa qualquer informação sobre recursos para agendamento de tarefas. Segundo Favarim, Fraga e Lung (2012), a primeira tarefa à espera de ser agendada é escolhida e um recurso livre é atribuído arbitrariamente para executá-lo. Este procedimento é repetido até que todas as tarefas programadas sejam executadas.

A ideia por trás do WQ é que mais tarefas serão atribuídas aos recursos mais rápidos, enquando os mais lentos irão processar uma carga de trabalho menor (BORRO, 2014).

2.5.6 Workqueue com replicação

Semelhante ao algoritmo WQ, *Workqueue* com o algoritmo de replicação, *Workqueue* com Replicação (WQR), também não usa qualquer informação, entretanto, segundo Favarim, Fraga e Lung (2012), quando não há mais tarefas a serem executadas e ainda existem recursos ociosos, as tarefas que ainda estão em execução são replicadas nestes recursos ociosos, ou seja, eles também são executados nos outros recursos. Quando uma tarefa replicada termina, todas as outras réplicas serão paradas.

A ideia desse algoritmo é a de que, quando uma tarefa é replicada, há uma possibilidade de que uma réplica seja atribuída a um nó mais rápido, aumentando assim a probabilidade de uma conclusão de forma mais rápida da tarefa.

Este algoritmo apresenta um bom desempenho quando há recursos suficientes para a replicação das tarefas (SILVA; CIRNE; BRASILEIRO, 2003).

2.5.7 Task grouping

O algoritmo *Task grouping* é indicado para escalonamento quando existe um grande número de tarefas consideradas de curta duração (GOMES et al., 2010).

Este algoritmo realiza o agrupamento das tarefas, de acordo com o seu tamanho de computação e a capacidade de processamento dos recursos da grade. Onde cada grupo de tarefas é enviado a um único recurso (GUADAGNIN, 2010).

Ainda segundo Guadagnin (2010), essa solução evita a existência de uma possível sobrecarga da grade caso as tarefas fossem escalonadas individualmente.

2.6 CONCLUSÃO

Este Capítulo apresentou o referencial conceitual no qual esta dissertação se fundamenta, o qual destaca à apresentar conceitos da Computação Verde e apresentando alguns algoritmos de escalonamentos que também sevirão como base para o desenvolvimento do algoritmo proposto por esta dissertação. O Próximo Capítulo apresenta os trabalhdos relacionados a este.

CAPÍTULO 3

TRABALHOS RELACIONADOS

Os trabalhos apresentados a seguir descrevem o escalonamento de tarefas e recursos usando heurísticas em sistemas distribuídos.

Yamini (2012) apresenta dois algoritmos de alocação de recursos: o Algoritmo *Energy Consolidation and Task Consolidation* (ECTC) e *Maximum Utilization* (MaxUtil). Ambas as heurísticas tem o objetivo de verificar todos os recursos disponíveis e identificarem a fonte de energia mais eficiente para realizar a tarefa. A diferença entre os algoritmos está nas duas heurísticas utilizadas que identificam os recursos disponíveis e a fonte mais eficiente. Entretanto, a solução apresentada nesse artigo é voltada para a computação em nuvem, diferente do proposto neste trabalho, que é voltado para as práticas da computação verde no paradigma da computação em grade. Yamini (2012) se preocupa apenas com informações sobre o consumo do processador.

Assim como Yamini (2012), o trabalho deWerner (2011) trata do problema do consumo de energia no ambiente da Computação Verde. É proposta uma estratégia para alocação e distribuição de máquinas virtuais em ambientes de Computação em Nuvem Verde, difenterente deste trabalho que busca a redução de energia em grades computacionais, entrentanto, Werner (2011) buca proporcionar um modelo de gerenciamento baseado em Teoria da Organização, com políticas de migração e realocação de recursos, para uma melhor eficiência energética na gestão de recursos. Ele busca realizar o gerenciamento dos recursos de acordo como nível de serviço, se baseando fundamentalmente na aplicação de critérios de provisionamento, alocação, redimensionamento e migração de máquinas virtuais para obter uma eficiente consolidação de carga nos servidores físicos.

Já Pinto (2013) apresenta uma solução para o problema de escalonamento de tarefas baseado em algoritmos genéticos para distribuir tarefas de maneira mais eficiente em uma grade computacional. Este trabalho teve como objetivo de reduzir o tempo total de execução das tarefas, permitindo assim, diminuir o tempo de processamento e aumentar a eficiência da grade. Entretanto, o mesmo não aborda a redução de consumo de energia.

O trabalho de Coutinho, Carvalho e Santana (2011) também é voltado para o ambiente em

grade computacional. Coutinho, Carvalho e Santana (2011) propõe uma heurística chamada de HGree, com o objetivo de redução de energia em problemas de programação de fluxo de trabalho[1] em um ambiente de Grade Computacional. Tal problema descrito consiste em atribuir tarefas de fluxo de trabalho para recursos da rede e definir a sua ordem de execução. Ainda segundo Coutinho, Carvalho e Santana (2011), o algoritmo HGreen visa identificar as tarefas que mais consomem energia, ou seja, quais são mais pesadas, distribuindo-as a recursos com maior eficiência energética. Consegue-se dessa forma, uma redução no consumo de energia.

Assim como o trabalho de Coutinho, Carvalho e Santana (2011), Silva C. (2012) também aborta o algoritmo HGreen, dando sequência e aprimorando ao mesmo. Este trabalho também apresenta mais dois algoritmos, o GGreen e o BOTEEN. O Trabalho de Silva C. (2012) está voltado para o ambiente de grades computacionais simulado no GridSim [2], limitando-se apenas a informações sobre o consumo de energia no processamento das tarefas.

O trabalho de Teodoro (2013), propõe um algoritmo de escalonamento de tarefas energeticamente eficiente para a gestão de consumo de energia em grades computacionais. A solução apresentada é baseada principalmente na gestão eficiente dos recursos ociosos e no uso inteligente dos recursos ativos.

O trabalho de Borro (2014) tem como objetivo o contexto do gerenciamento energético em grades móveis, onde o mesmo propõe dois algoritmos de escalonamentos (*Maximum Regret* e *Greedy*) que buscam minimizar o consumo de energia, considerando cenários estáticos e dinâmicos. Tais algoritmos foram projetos a partir de soluções heurísticas para o problema de escalonamento ciente de consumo de energia em grades móveis. Entretanto, o trabalho do mesmo não se preocupa com dispositivos conectados a uma fonte de energia.

A tabala 3.1 apresenta as diferentes entre este trabalho e os trabalhos acima citados.

A solução que será apresentada nesta dissertação tem como objetivo proporcionar a redução do consumo de energia da grade computacional, permitindo assim, a diminuição dos efeitos de degradação do meio ambiente, sem comprometer a eficiência computacional.

Entretanto, a principal contribuição deste trabalho em relação aos trabalhos citados é a heu-

[1] Um fluxo de trabalho pode ser visto como um conjunto de tarefas computacionais que são processadas conforme uma ordem pré-definida

[2] o Gridsim é uma ferramenta de simulação que suporta modelagem e simulação de eventos discretos ede arquiteturas em grade heterogêneas, tal como mono ou multiprocessadores, máquinas de memória compartilhada e distribuída, e escalonamento de tarefas em sistemas paralelos e distribuídos (aglomerados de computadores etc.)

Tabela 3.1: Comparação entre trabalhos apresentados

Trabalho	**Ambientel**	**Algoritmo de Escalonamento**	**Verde?**
YAMINI, 2012	Computação em Nuvem	ECTC e MaxUtil	Sim
PINTO, 2013	Grid	Algoritmo Genético	Não
WERNER, 2011	Computação em Nuvem	Teoria da Orgazinação	Sim
COUTINHO; CARVALHO; SANTANA, 2011	Grid	HGreen	Sim
SILVA C., 2012	Grid	HGreen, GGreen e BOTEEN	Sim
TEODORO, 2013	Grid	LECSA e DLECSA	Sim
BORRO, 2014	Grid Móveis	Maximum Regret e Greedy	Sim
Este Trabalho	**Grid**	**GVerde**	**Sim**

rística a qual é usada na implementação do algoritmo de escalonamento de recursos e tarefas. Também serão realizados diversos testes e comparações com algoritmos de escalonamento em máquinas físicas, diferente dos trabalhos citados anteriormente, onde os mesmos foram simulados.

É importante ressaltar também que os trabalhos anteriormente citados por serem simulados e por todos trabalharem com poucos componentes de computadores, a maioria dos casos apenas se preocupa com o processamento, portanto, apenas com o consumo de energia do processador, deixando de lado outros componentes de um computador físico que também tem importância vital no consumo de energia (tais como monitor, memória, barramento, entre outros componentes), diferente deste trabalho que aborda o consumo de um nodo físico, logo, o consumo do mesmo com todos os seus componentes. Entrentanto, por serem simulados, apresentantam resultados com grande quantidades de recursos e tarefas, diferente desta dissertação, que apresenta esta limitação no número de recursos.

3.1 CONCLUSÃO

Este Capítulo apresentou os trabalhos relacionados a este, destacando as principais características dos trabalhos relacionados e também destacando as diferenças mais importantes em

comparação a esta dissertação. Também é abordado as principais contribuições desta dissertação em relação aos trabalhos citados. O Capítulo seguinte apresenta o Modelo Energético adotado por esta dissertação.

CAPÍTULO 4

MODELO DE ENERGIA

Um modelo energetico pode ser dividido em duas categorias, conforme descrito em Rivoire, Ranganathan e Kozyrakis (2008): modelos completos (*comprehensive models*) e modelos de alto nível chamados de caixa-preta (*black-box*). O primeiro tipo permite obter grande acuracia do consumo energetico dos computadores, entretanto, é dependente dos detalhes de cada componente. O segundo tipo de modelo descreve de forma generica o consumo energético. Não sendo tão preciso e permitindo ampliar sua portabilidade, não necessitando de detalhes específicos das arquiteturas utilizadas (WATANABE et al., 2014).

A energia consumida por um sistema computacionais, como uma grade computacional, abrange diversos componentes, tais como citado por Silva C. (2012): iluminação, refrigeração, redes e elementos computacionais. Entretanto, a redução de consumo que esta dissertação trata é apenas elementos computacionais, ou seja, apenas informações referentes ao consumo dos recursos durante os estados ociosos (em espera) e de execução dos mesmos são consideradas.

Logo, o modelo energético apresentado nesta dissertação se baseia no modelo de energia apresentado no trabalho de Yamini (2012), onde o mesmo tem como base a utilização do processador em uma relação linear com o consumo de energia.

Este modelo tem como objetivo estimar o consumo na execução de uma aplicação e, posteriormente, saber quanta energia gasta por cada algoritmo de escalonamento durante os testes desta dissertação.

Entretanto, este trabalho visa o consumo energético de todos componente do nodo, logo, não são avaliados apenas o consumo do processador, mas também de todos os componentes que compõe os nodos da grade computacional.

Para obter o consumo de energia, tem-se:

$$E = (pmax * tu) + (pmin * to) \tag{4.1}$$

Sendo "*pmax*" a energia consumida na execução do recurso e "*pmin*" a energia consumida

no estado ocioso. Já "*tu*" é o tempo de utilização do recurso e "*to*" é tempo do recurso no estado ocioso.

O consumo total de energia é dado pela soma de todos os recursos da grade. Sendo "*n*" o número de recursos, temos:

$$ConsumoTotal = \sum_{i=1}^{n} E_i \tag{4.2}$$

Portanto, o consumo total de energia pela grade é dado pela soma dos consumos de cada recurso.

4.1 FERRAMENTA PARA ESTIMAÇÃO DE CONSUMO DE ENERGIA

Existe poucas ferramentas para a medição do consumo de energia em computadores. Para calcular o consumo de energia em dispositivos de hardware, em geral, oferecem poucas ferramentas para medir o seu consumo energético (SALGADO, 2011). Entretanto, segundo Silva J. (2012), existem softwares específicos que realizam uma avaliação de consumo a partir de informações obtidas pelo sistema operacional de um dado dispositivo computacional.

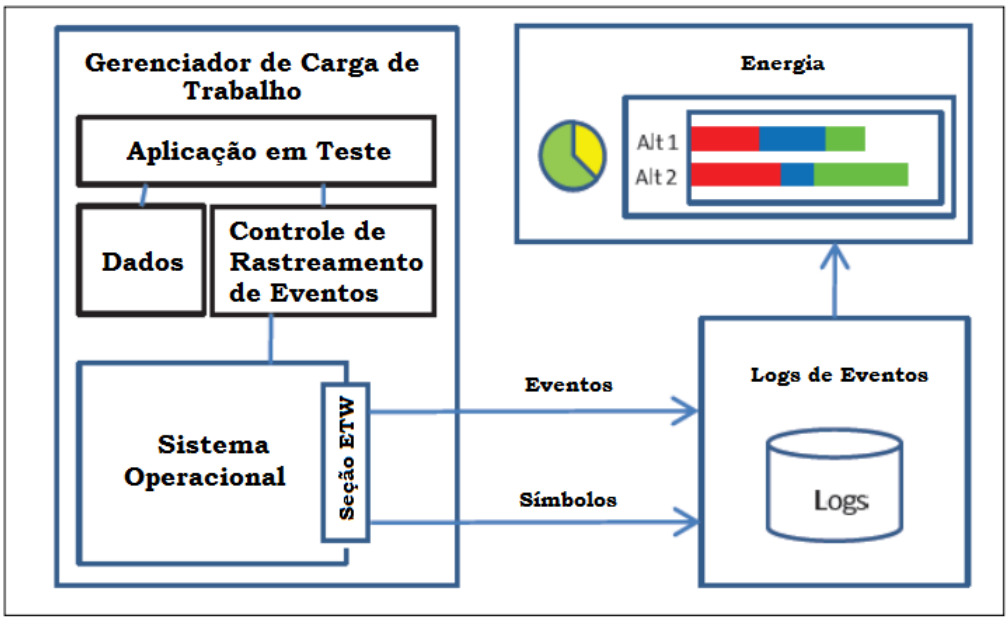

Figura 4.1: Esquema geral do Joulemeter (GUDE, 2010).

Nesta dissertação o consumo de energia será estimado, em cada componente da grade implementada, através da ferramenta *Joulemeter*, qual é desenvolvida pela Microsoft (GORACZKO

et al., 2011). Ela é capaz de estimar o consumo de energia de um computador através do rastreamento de informações sobre diversos componentes de hardwares (ORGERIE; ASSUNCAO; LEFEVE, 2014). O Joulemeter é capaz de monitorar o consumo em Máquina Virtuais (MVs), servidores, desktops, laptops e componentes de software (GORACZKO et al., 2011).

Entretanto, é propício ressaltar que, quando se necessita de exatidão e confiabilidade nas medidas de consumo energético deve-se utilizar instrumentos eletrônicos específicos.

A ferramenta Joulemeter é mostrada esquematicamente na Figura 4.1, e é composta por três componentes principais: gerenciador de carga de trabalho, registro de eventos e de perfil de energia.

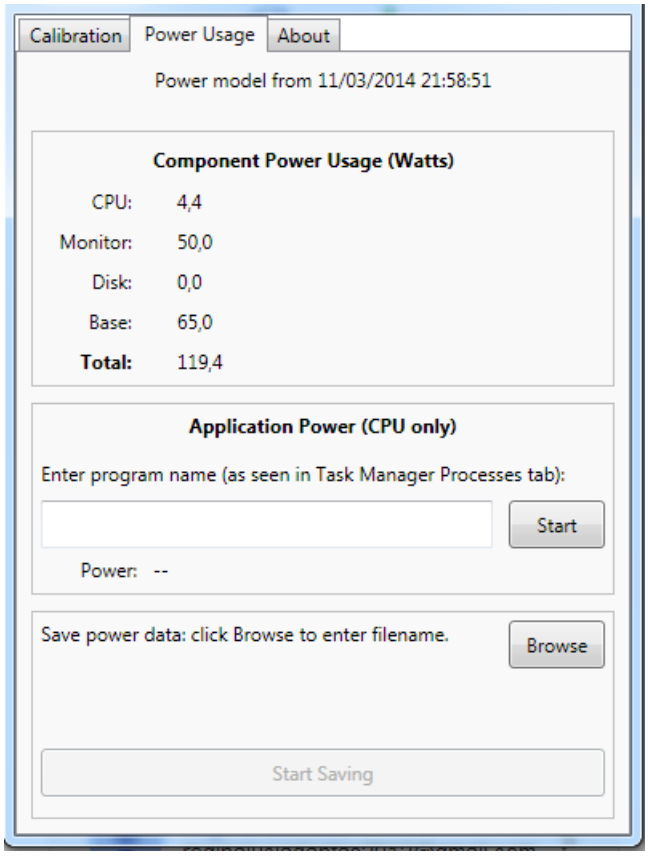

Figura 4.2: Joulemeter em execução.

O gerente de carga de trabalho gerencia o rastreamento de eventos e executa o programa com alguma carga de trabalho de dados. Ele usa Rastreamento de Eventos do Windows (REW) (*Event Tracing for Windows (ETW)*) para o sistema de rastreamento de nível de kernel para a geração de eventos que podem ser registrados pelo registrador de eventos. O registrador de eventos usa *Xperf* para isso. O arquivo de energia analisa o log final em texto legível e gráfico

(GUDE, 2010).

A Figura 4.2 apresenta o Joulemeter em execução em um dos computadores utilizado nos testes deste trabalho.

Um problema encontrado com o uso do Joulemeter é que o mesmo é desenvolvido para o ambiente do Windows, impossibilitando o uso do mesmo em outros sistemas operacionais.

4.2 CONCLUSÃO

Este Capítulo apresentou o Modelo de Energia adotado por esta dissertação, destacando a forma como é feito o cálculo para obter o consumo de energia da grade computacional implementada nesta. Também é apresentado a ferramenta Joulemeter, qual é usada para obter o consumo individual dos nodos. O Capítulo seguinte apresenta o algoritmo implementado.

CAPÍTULO 5

O ALGORITMO DE ESCALONAMENTO VERDE PROPOSTO

Algoritmos de escalonamentos de recursos em grids computacionais levam em conta diversos aspectos, como citado por Silva C. (2012): CPU, memória, disco, largura da banda, histórico de execuções anteriores, entre outros.

Nessa dissertação é considerado o requisito de eficiência energética, buscando a redução do consumo de energia utilizada pela grade computacional e não dos nodos de forma individual, reduzindo através de políticas de computação verde, entretanto, sem comprometer o desempenho computacional e mantendo o sistema eficaz.

O algoritmo proposto neste trabalho se trata do GVerde (”GRASP Verde”), o qual é um algoritmo baseado no método *Greedy Randomized Adaptive Search Procedure* (GRASP) (FEO; RESENDE, 1989) para soluções de escalonamentos de recursos e tarefas em grades computacionais.

O GRASP foi proposto por Feo e Resende (1989) como um método iterativo constituído de duas fases: a de construção e a de busca local. A Figura 5.1 aprensenta o pseudo-código do GRASP. O GRASP gera uma solução inicial de melhor qualidade para ser utilizado na busca local, que realiaza apenas pequenas melhorias.

```
Enquanto (condição de parada não for satisfeita), Faça
   solução ← crie aleatoriamente uma solução de forma construtiva();
   solução ← busca local(solução);
   Se solução é a melhor solução até Então conhecida então
      grave(solução);
   Fim Se
Fim Enquanto
```

Figura 5.1: Pseudo-código do GRASP

Na maioria das aplicações, o critério de parada é baseado no número máximo de iterações. Podem-se definir outros critérios, como por exemplo parar quando a solução procurada for encontrada ou estabelecer um tempo máximo de execução (RANGE; ABREU; BOAVENTURA-NETTO, 2000).

Uma vez gerada uma nova solução através da busca local é testado se está nova solução apresenta resultados melhores valores do que a anterior, apresentando melhorias viáveis para o problema.

Relacionado com esta questão do impacto mínimo ao meio ambiente, o algoritmo GVerde apresentado neste trabalho, toda a sua lógica é direcionada para reduzir o consumo de energia em um ambiente heterogêneo de um sistema distribuído, entretanto, sem comprometer o mínimo possível do desempenho computacional obtido por sistemas da grade computacional em termos de tempo de execução.

O GVerde é dividido em três partes, onde, a primeira parte busca criar uma sequência de execução de forma eficiente energeticamente (Solução Inicial). Já a segunda é destinada a executar alterações na solução criada com o objetivo de melhorá-la (Busca Local). A terceira parte, procura evitar que nodos fiquem ociosos enquanto existir tarefas à serem executadas. O GVerde é apresentado estruturamente pela Figura 5.2;

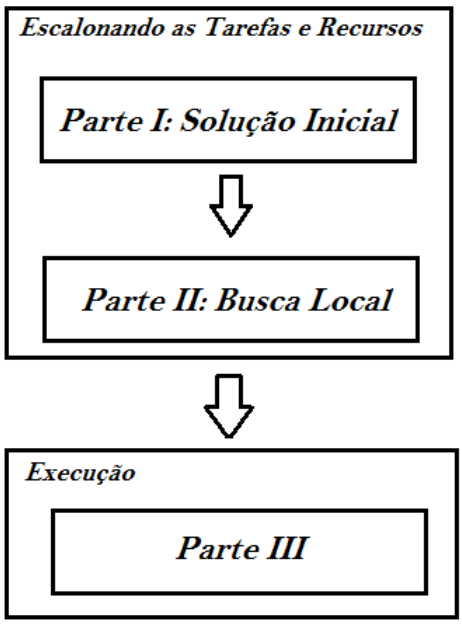

Figura 5.2: Estrutura do GVerde

Na Parte I, a solução inicial é gerada seguindo alguns critérios, tais como carga da tarefa e consumo energético dos recursos, para buscar reduzir o consumo de energia.

Inicialmente as tarefas são ordenadas em ordem decrescente de acordo com o seu tamanho das cargas destas tarefas e os recursos são ordenados também de forma decrescente de acordo com o seu consumo de energia.

No passo seguinte é gerada uma sequência de execução procurando dividir as tarefas em quantidades iguais em cada nodo.

O segundo passo da Parte I, consiste na ideia de criar uma lista de execução onde as tarefas mais pesadas executem em primeiro instante, logo, deixando as tarefas com menor peso de carga para o final.

```
     Entrada: ListaTarefas e ListaRecursos;
     Saída: SoluçãoInicial;

1    Lista SoluçãoInicial;
2    Inteiro t ← 1;
3    Execução execução;

4    ListaDeTarefas ← ordenarTarefasPorPeso(ListaTarefas);
5    Para Cada i Até ListaRarefas.tamanho() Faça
6        Se t > ListaRecursos.tamanho() Então
7            t ← 1;
8        Fim Se;
9        execução ← novaExecução(ListaTarefas(i), ListaRecursos(t));
10       SoluçãoInicial.adicional(Execução);
11       t++;
12   Fim Para;
13   Retorne SoluçãoInicial;
```

Figura 5.3: Parte I do GVerde

Como se tem na Figura 5.3, na linha 1 é criado uma lista que irá armazenar a Solução Inicial (SoluçãoInicial). Já na linha 2 é iniciado um inteiro "*t*", qual será usado como um contador. Na linha 3 é criado execução do tipo Execução[1];

Na linha 4, a lista de tarefas é ordenada por peso[2]. Quanto maior a carga de trabalho (tamanho do vetor), maior será o seu peso.

Da linha 5 até a linha 12 é implementado um laço "*Para*" com o objetivo de dividir as tarefas entre os nodos da grade computacional. Onde na linha 6 sempre testa se o inteiro "*t*" é maior do que a lista de recursos, em caso de verdadeiro, o "*t*" recebe 1 (um). Se falso, não faz nada e

[1] Execução foi implementado como um tipo que armazena o código da tarefa e o código do recurso
[2] Este peso é defino anteriormente e manualmente, de forma tabela para cada tarefa

segue para a linha 9 onde é criada uma nova execução, onde "*execução*" recebe a tarefa "*t*" e o recurso "*t*".

Após na linha 10, esta execução é inserida na lista "*SoluçãoInicial*". Já na linha 11 "*t*" é incrementado. Este laça quanto dadas as tarefas tiverem sido destinadas a algum recurso da grade. Na linha 13, o método retorna a SoluçãoInicial gerada pelo método.

É importante destacar que a Parte I buscar criar uma solução inicial de forma simples sem consumir muito tempo, já que quanto mais tempo demorar a Parte I, mais tempo os recursos passarão ociosos e consumindo energia. Quanto maior o número de tarefas e recurso, maior será o tempo para gerar a solução inicial.

Na segunda parte acontece a busca local, cujo objetivo é melhorar a solução inicial gerada anteriormente. Nesta fase, são realizadas alterações na solução de forma aleatória, escolhendo uma tarefa qualquer e depois escolhendo um nodo onde apresente o menor consumo na execução da tarefa. A Figura 5.3 mostra o pseudo-código referente a parte 2 do algoritmo GVerde.

```
    Entrada: SoluçãoInicial;
    Saída: SoluçãoFinal;

1   Lista ListaSoluçãoTemp ← SoluçãoInicial;
2   Lista SoluçãoFinal ← SoluçãoInicial;

3   Enquanto (condição de parada não for satisfeita), Faça
4       Execução execução ← aleatório(SoluçãoTemp);
5       Tarefa t ← execução.pegarTarefa();
6       Recurso r ← escolherRecursoComMenorConsumo(t);
7       execução.atualize(t,r);
8       SoluçãoTemp.atualize(execução);
9       Se consumo(SoluçãoInicial) > consumo(SoluçãoTemp), Então
10          SoluçãoFinal ← SoluçãoTemp;
11      Senão, Faça
12          SoluçãoTemp ← SoluçãoFinal;
13      Fim Se
14      Fim Enquanto
15      Retorne SoluçãoFinal;
```

Figura 5.4: Parte II do GVerde

A segunda parte do GVerde refere-se a um método que recebe como entrada uma Solução Inicial gerada anteriormente pela Parte I e tem como saída uma lista de Solução Final (lista de execução).

Na linha 1 é criada uma solução temporária ("*ListaSoluçãoTemp*"), a qual recebe "*SoluçãoInicial*". Já na linha 2, "*SoluçãoFinal*" também recebe "*SoluçãoInicial*".

Na linha 3 inicia-se o laço *Enquanto*, que termina na linha 14. Enquanto a condição de parada não acontecer, será realizado a busca local para melhorar a "*SoluçãoFinal*" salva. Foi adotado nos teste que o *Enquanto* executaria metade do número de tarefas submetidas.

Dentro do laço *Enquanto* na linha 4 é escolhido uma execução qualquer da lista "*SoluçãoTemp*. Nas linhas 5 seleciona a tarefa e referente a execução selecionada na linha anterior.

Na linha 6, é escolhido um recurso disponível que apresente menor consumo para a execução da tarefa anteriormente escolhida.

Nas linhas 7 e 8 é feito as atualizações respectivamente: da "*execução* (associando a tarefa "*t*" com o recurso anteriormente selecionado) e a "*SoluçãoTemp* com a execução com o novo recurso.

Seguinte o algoritmo entra em um "Se", onde é testado se o consumo de energia da "*SoluçãoFinal*" é maior do que o consumo da solução alterada ("*SoluçãoTemp*"). Em caso de verdadeiro a "*SoluçãoFinal*" recebe a solução alterada. Em caso de falso, a "*SoluçãoTemp*" é desfeita, onde a mesma recebe a "*SoluçãoFinal*".

Quando a condição de parada do "*Enquanto*" for satisfeita, é retornado a "*SoluçãoFinal*" na linha 15.

A ideia da Parte II consiste em atribuir as tarefas aos recursos onde elas possuem menor consumo de energia, permitindo assim a redução do consumo pela grade.

Quando um nodo concluir todas as execuções das suas tarefas, a tendência é ele ficar no estado ocioso, consumindo energia, entretanto, a Parte III busca evitar que o mesmo fique neste estado, consumindo energia, logo, é destinado uma tarefa do final da lista para este nodo, a fim de evitar o estado ocioso, além de tornar a execução do grid mais eficiente.

Quando um nodo concluir todas as suas tarefas destinadas, o mesmo é inserido em uma lista de recursos disponíveis. Durante a execução das tarefas, o Algoritmo GVerde sempre fica consultando esta lista em busca de nodos que concluíram as suas tarefas e quando existe um

```
1   Se (ListaDeRecursosDisponíveis > 0 ), Então
2       Recurso r ← ListaDeRecursosDisponíveis.pegarRecurso()
3       Tarefa f ← execução.pegarÚltimaTarefaNãoExecutada();
4       Execução.remova(t);
5       ListaDeRecursosDisponíveis.removaRecurso(r);
6       execute(r,t);
7   Fim Se
```

Figura 5.5: Parte III do GVerde

ou mais nodos nesta lista, é destino uma nova tarefa para este nodo. A Figura 5.5 apresenta o pseudo-código referente a Parte III do GVerde.

A terceira parte do GV é basicamente um "*Se*" que testa se exista algum recurso disponível na grade computacional. Quase existe uma ou mais recurso (verdadeiro) o GVerde seleciona de forma aleatória da lista de recursos disponível e pega a última tarefa da lista de execução (linha 2). Depois na linha 3 pega a última tarefa da lista de execução ainda não executada.

Na linha 4 esta tarefa selecionada é removida da lista de execução, com o intuito de não ser executada novamente por outro nodo futuramente. O Algoritmo também remove o nodo selecionado da lista dos recursos disponíveis (linha 5). Depois é realizado a execução da respectiva tarefa pelo nodo selecionado anteriormente (linha 6).

O GVerde seleciona esse recurso disponível e pega a última tarefa da lista de execução, então, esta tarefa é retirada da lista de execução e o nodo da lista dos recursos disponíveis. Logo em segunda é realizada a execução da respectiva tarefa pelo nodo selecionado anteriormente.

O GVerde com estes passos descritos é capaz de reduzir de forma significativa o consumo de energia de uma grade computacional, como poderá ser visto no Capítulo seguinte, destinado para Testes e Resultados.

5.1 CONCLUSÃO

Este Capítulo apresentou o algoritmo GVerde, qual é baseado no GRASP com o foco no escalonamento de recursos e tarefas e com o objetivo de proporcionar a redução do consumo de energia em uma ambiente de grades computacionais. É descrito as principais características do GVerde, destacando principalmente as características da computação verde presente neste

algoritmo. É descrito o pseudo-código do GVerde, destacando as trrês partes que compõe o algoritmo proposto e foi comentado as principais linhas do pseudo-código. No próximo Capítulo é abordado os testes e resultados obtidos com o GVerde.

CAPÍTULO 6

RESULTADOS OBTIDOS

O objetivo deste capítulo é de avaliar e analisar o desempenho e o consumo de energia do algoritmo desenvolvido nesta dissertação em relação a outros algoritmos de escalonamento para grades computacionais já existentes.

Primeiramente, foram realizados testes com os nodos para obter uma média de tempo de execução de cada tarefa em cada um dos nodos. Médias estas que serão usadas com informação para os algoritmos GRASP e GVerde. Também foram realizados testes com os nodos para obter o consumo de energia dos mesmos.

Em relação a validação dos resultados, foram realizados testes também com o algoritmo Workqueue e o GRASP, permitindo assim, mostrar a eficiência do algoritmo proposto. Segundo (BORRO, 2014), O algoritmo Workqueue não necessita de informações sobre as tarefas e/ou recursos, o mesmo realiza o escalonamento aleatoriamente, alocando-se uma tarefa qualquer a um recurso disponível. Já o GRASP implementado para testes nesta dissertação não utiliza informação sobre eficiência energética dos recursos, apenas informações sobre tempo de processamento dos mesmos.

Para a realização dos testes, foram utilizado dois módulos de testes, com diferentes níveis de heterogeneidade entre as máquinas.

Nas Seções que seguem, apresenta-se o ambiente e os cenários de testes utilizados, e por fim, será feito uma análise dos resultados obtidos durante a realização dos testes.

6.1 AMBIENTE DE TESTE

Para o desenvolvimento do ambiente de teste, foi utilizado a linguagem de programação JAVA para a implementação da grade computacional, o escalonador, tarefas, entre outros.

A Figura 6.1 apresenta a estruturação da grade computacional utilizada para os testes, onde o mesmo é formado por um Nodo de Controle, que possui todos os recursos da grade, diferenciando dos outros nodos apenas por possuir característica de recursos administrativos, tais como a reponsabilidade pelo escalonamento e por possuir os recursos de informações, no caso desta

grade, o *Grid Information System* (GIS).

Para acessar os componentes internos do nodo de controle, utiliza métodos internos, como por exemplo para acessar o GIS. Já para realizar a comunicação com outros nodos, foi implementados serviços de XML-RPC [1], para a realização da comunicação. É através de chamadas XML-RPC que os outros nodos podem obter informações ao GIS.

O XML-RPC é basicamente formado por elementos XML e por um cabeçalho HTTP para o envio de chamadas a métodos ou a procedimentos pela rede de computadores feitos por clientes e servidores (SILVA; MARTINO, 2007).

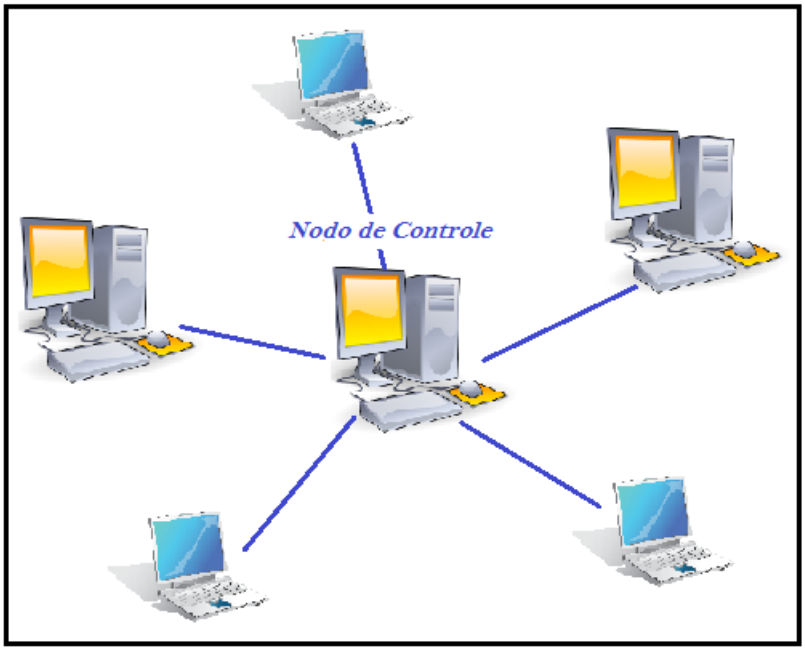

Figura 6.1: Estruturação da grade computacional implementado para a realização dos testes

A grade computacional é formada por vários nodos com diferentes características, em diversos níveis de heterogeneidade entre os mesmos, desde poder de processamento ao nível do consumo de energia. A Tabela 6.1 mostra as principais características dos nodos, destaque para os nodos 2 e 5, que são os que mais consomem energia na grade computacional, possuindo uma grande diferença no consumo em relação ao outros nodos que compõe a grade de teste[2].

Para determinar o consumo de energia dos nodos foi utilizada a ferramenta Joulemeter (GORACZKO et al., 2011). Segundo Orgerie, Assuncao e Lefeve (2014), essa ferramenta é capaz de

[1] XML-RPC é um protocolo de chamada de procedimento remoto (*Remote procedure Call* (RPC)), que usa *Extensible Markup Language* (XML) para codificar suas chamadas e *Hypertext Transfer Protocol* (HTTP) como mecanismo de transporte.

[2] A grade computacional utilizada no teste foi implementado com esses 6 nodos, sendo os nodos 2 e 5 computadores do tipo Desktop, já os outros, 1, 3, 4, 6 são notebooks

Tabela 6.1: Principais Características dos Nodos usados na simulação

Nodos	**CPU**	**RAM**	**Consumo em Watts**	
			Máximo	**Mínimo**
Nodo 1	Intel Core i3 2.10 GHz	4 GB	22	19
Nodo 2	Intel Core 2 Duo 2.93 GHz	4 GB	135	120
Nodo 3	Intel Pentium Dual 2.16 GHz	2 GB	32	22
Nodo 4	Intel Celeron Dual Core T3100 1.90 GHz	2 GB	29	24
Nodo 5	Intel Core i7 3.40 GHz	6 GB	120	115
Nodo 6	Intel Celeron 1007U 1.50 GHz	2 GB	29	26

estimar o consumo de energia de um computador através do rastreamento de informações sobre diversos componentes de hardwares.

6.2 AS TAREFAS

Para realizar os testes foram desenvolvidos tarefas de ordenação de vetores utilizando os algoritmos de inserção e de seleção, modificando apenas o tamanho dos vetores a serem ordenados, permitindo assim diferentes cargas para os testes. Segue a descrições da tarefas:

- **Tarefa 1**: método de seleção para ordenação de um vetor de 10.000 elementos;
- **Tarefa 2**: método de seleção para ordenação de um vetor de 50.000 elementos;
- **Tarefa 3**: método de seleção para ordenação de um vetor de 100.000 elementos;
- **Tarefa 4**: método de inserção para ordenação de um vetor de 10.000 elementos;
- **Tarefa 5**: método de inserção para ordenação de um vetor de 50.000 elementos;
- **Tarefa 6**: método de inserção para ordenação de um vetor de 100.000 elementos;

O método de ordenação por seleção é baseado a ideia de escolher um menor elemento do vetor, depois um segundo menor elemento e assim por diante. O Tempo de execução do pior caso para este algoritmo é n^2 (MARTINEZ, 2011).

O algoritmo de ordenação por inserção é muito popular. Ele é frequentemente usado para colocar em ordem um baralho de cartas (FEOFILOFF, 2009). Entretanto, nesta dissertação é usado para ordenar um vetor de números inteiros. Ainda segundo Feofiloff (2009), o tempo do

método de inserção em pior caso é proporcional a n^2. Este algoritmo é lento quando n é muito grande.

Para a realização dos testes as tarefas são geradas de forma aleatória, não obedecendo nenhuma ordem de sequência ou de seleção das mesmas.

6.3 MÓDULO 1: GRADE COMPUTACIONAL FORMADA COM 4 NODOS

Para a realização dos testes referente a esta Seção, foram escolhidos os 4 nodos de acordo com a Tabela 6.1, no caso, os nodos 1, 3, 4 e 6. A ideia deste Módulo de teste é de buscar obter o consumo em máquinas que apresente uma heterogeneidade baixa em relação ao consumo energético dos mesmos, então, dos nodos disponíveis, foram escolhidos estes 4 nodos, por apresentarem um consumo de energia semelhantes.

A Tabela 6.2 mostra os resultado obtidos com os testes referente ao Módulo 1, onde é apresentado o tempo de duração médio (em segundos) e o consumo energético médio (em *Watts*) para executar um determinado conjunto de tarefas obtido em ambos os algoritmos de escalonamento.

Tabela 6.2: Resultados dos testes no Módulo 1

Quant. Tarefas	Workqueue		GRASP		GVerde	
	s	W	s	W	s	W
100	167,33	18420,67	167,33	18471,33	167,33	18365,00
150	234,33	25918,00	222,67	24652,67	175,00	24322,33
200	302,67	33570,33	295,67	32678,33	262,67	28981,33
300	456,00	50555,00	450,00	49992,67	402,33	43889,00
450	683,33	75950,00	670,00	74409,33	588,33	65106,33
600	892,00	99081,67	860,33	95610,33	761,00	84310,67

Como visto na Tabela 6.2 e na Figura 6.2, o algoritmo GVerde apresentou o melhor resultado em relação ao consumo de energia comparado com os outros algoritmos quando testado com 600 tarefas, chegando apresentar uma eficiência energética de em média 15% em relação ao Workqueue e 12% em relação ao GRASP. Já no teste com 100 tarefas apresentou a menor redução de energia, correspondendo apenas a uma média de 0,30% em relação ao Workqueue e 0,60% em relação ao GRASP.

O GVerde teve um total médio de redução de consumo energético de 10,5% em relação ao algoritmo Workqueue. Já em relação ao GRASP, a média total do consumo foi de 8,3%.

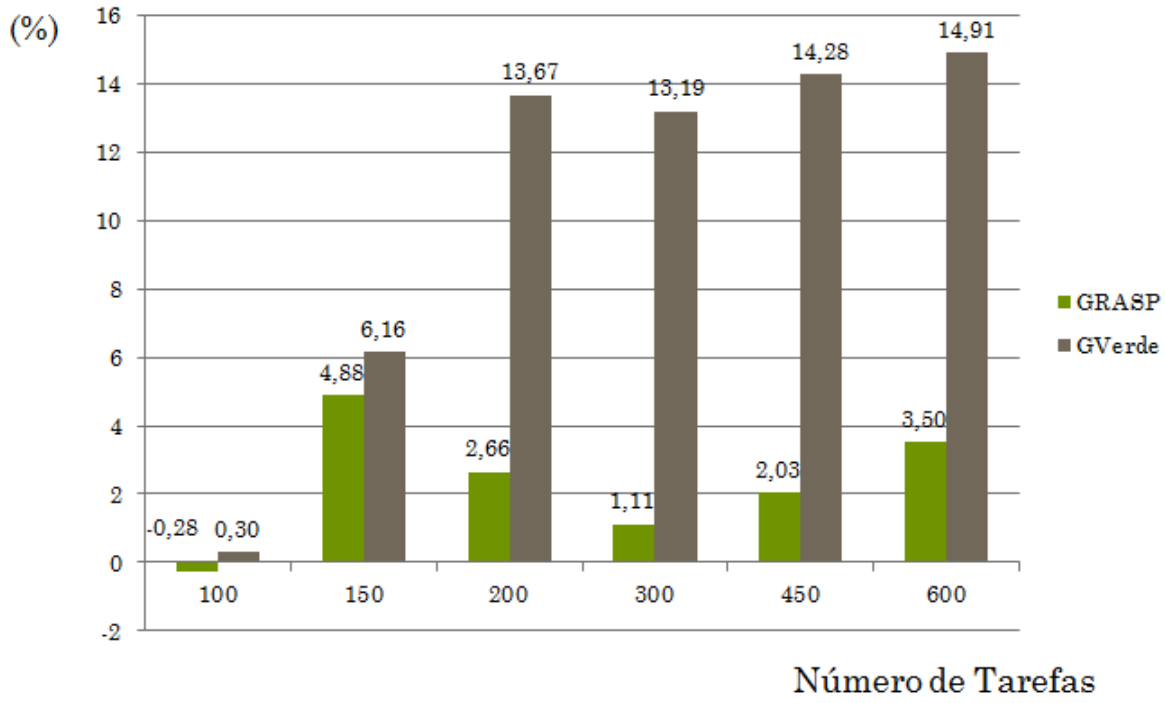

Figura 6.2: Redução do consumo de energia em relação ao Workqueue

Neste Módulo de teste, o algoritmo GRASP apresentou uma eficiência energética semelhante ao Workqueue, onde esta redução teve uma média de 2,32%, sendo que no teste com 100 tarefas, apresentou maior consumo de energia entre os três algoritmos testados, como pode ser visto na Tabela 6.2.

6.4 MÓDULO 2: GRADE COMPUTACIONAL FORMADA COM 6 NODOS

No segundo Módulo de Teste, foram adicionados mais dois nodos, esses apresentando maior consumo energético em relação aos outros nodos do Módulo anterior, como visto na tabela 6.1, permitindo assim uma maior heterogeneidade da grade computacional.

Analisado a Tabela 6.3, pode-se observar que o algoritmo GVerde apresentou o seu melhor resultado comparativos quando foi testado com 450 tarefas, chegando a uma redução do consumo de energia de 16,22% em relação ao algoritmo Workqueue e 12,74% em relação ao algoritmo GRASP.

O pior desempenho do GVerde foi quando testado com 300 tarefas, onde apresentou uma eficiência energética superior de 11,52% em relação ao Workqueue. Entretanto, consumiu 9,01% a menos energia do que o algoritmo GRASP no teste com este número de tarefas como pode ser observado na Figura 6.3.

Tabela 6.3: Resultados dos testes no Módulo 2

Quant. Tarefas	Workqueue		GRASP		GVerde	
	s	W	s	W	s	W
100	105,00	37925,25	95,00	34313,25	90,75	32681,25
150	135,25	49091,75	122,50	44389,00	117,75	42590,00
200	182,25	66059,00	170,00	61675,75	158,00	57050,75
300	259,75	94430,25	252,25	91823,75	229,75	83548,75
450	402,50	146447,00	386,50	140606,00	312,25	122694,50
600	509,75	185646,25	494,25	180056,75	440,75	160459,75

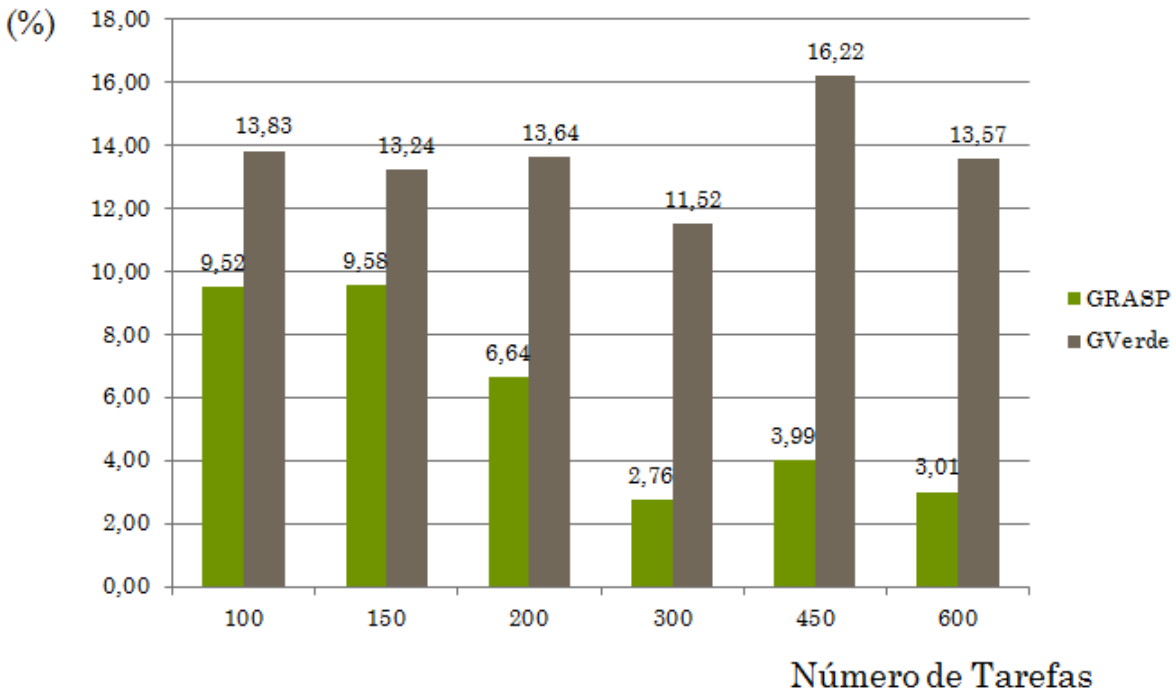

Figura 6.3: Redução do consumo de energia em relação ao Workqueue

Já o GRASP apresentou a característica de quanto mais tarefas, menor foi a sua eficiência energética em relação ao algoritmo Workqueue. Entretanto o algoritmo proposto por esta dissertação, mantem uma média, não apresentando uma grande variação em relação ao número de tarefas.

6.5 COMPARAÇÃO: MÓDULO 1 X MÓDULO 2

Quando comparados os Módulos, o Módulo 2 apresenta maior consumo de energia ocasionado pelo acréscimo dos dois nodos (nodo 2 e 5), entretanto, o tempo de execução de todas as tarefas na grade computacional diminuiu. Como é apresentado na Tabela 6.4, onde é apresentado o

somatório dos tempos e consumo de energia dos testes referente aos Módulos.

Tabela 6.4: Somatório dos tempos e consumo de energia dos testes referente aos Módulos

Algoritmo	**Módulo 1**		**Módulo 2**	
	s	**W**	**s**	**W**
Workqueue	2735,67	303495,67	1594,50	579599,50
GRASP	2666,00	295814,67	1520,50	552864,50
GVerde	2356,67	264974,67	1349,25	499025,00

Portanto, de acordos com os resultados dos dois Módulos:

- O algoritmo Workqueue apresentou menor tempo de execução no Módulo 2, chegado a 41,71%. Já em relação ao consumo de energia, no Módulo 1 apresentou uma redução do consumo de energia de 47,64%;
- Já o algoritmo GRASP apresenta no Módulo 2 uma eficiência de desempenho no tempo computacional de 43,97% em relação ao tempo de execução do Módulo 1. Mas quando comparado o consumo energético, o Módulo 1 apresentou uma eficiência energética de 46,49% em relação ao Módulo 2;
- Quando comparado entre os Módulos o desempenho do tempo de execução do GVerde, o Módulo 2 apresentou menor tempo, sendo 42,75% mais eficiente em relação ao tempo de execução do Módulo 1. Já em quando comparado o consumo energético, o Módulo 1 apresentou maior eficiência energética, chegando a 46,90% em relação ao segundo Módulo.

6.6 CONCLUSÃO

Este Capítulo apresentou testes e resultados com o objetivo de validar o algoritmo proposto nesta dissertação, no caso, o algoritmo GVerde. Também foi apresentado a descrição do ambiente de teste e descritos as tarefas. Depois foi abordado dois módulos de ambientes de testes para esta dissertação. Depois foi realizado comparações entre os resultados obtidos nos módulos. O Capítulo seguinte é destinado as conclusões desta dissertação e sugestões para trabalhos futuros.

CAPÍTULO 7

CONCLUSÕES E TRABALHOS FUTUROS

Na presente dissertação foi apresentada uma proposta de algoritmo de escalonamento de recursos e tarefas em um ambiente de grades computacionais. O presente capítulo revisa o que foi apresentando ao longo dissertação.

7.1 CONSIDERAÇÕES FINAIS

Como ponto de partida para o desenvolvimento dissertação, foi realizado um estudo referente à Computação Verde, abortando suas principais características e suas técnicas, qual serviu como base para a implementação do algoritmo de escalonamento verde, voltado para a eficiência energética em ambientes de grades computacionais.

Em seguinte, este trabalho teve como principal objetivo propor um algoritmo de escalonamento baseado no GRASP capaz de ser energeticamente eficiente, sendo desenvolvido então o algoritmo GVerde. O algoritmo utilizou técnicas da computação verde, permitindo assim uma maior eficiência energética como um todo em uma grade computacional sem comprometer o desempenho computacional, proporcionando diversos benefícios ao meio ambiente, tais como menor consumo de energia.

O GVerde pode proporcionar êxito durante os testes, principalmente na redução do consumo de energia da grade computacional implementada. Em alguns casos, essa redução chegou a superar a média de 16%, já no pior caso, chegou a uma redução de 0,3% comparado com os outros algoritmos.

7.2 TRABALHOS FUTUROS

Existem três sugestões para trabalhos futuros, a primeira é implementação de outras heurísticas baseadas em computação verde, com o objetivo de diminuir o impacto ambiental sem, no entanto, comprometer os processos computacionais.

A segunda trata de melhorar o algoritmo GVerde, por exemplo, aplicar outras técnicas da computação verde na primeira fase, tais como a redução de freuência do processador de um

recurso ou ainda desligamento de recursos não mais usados, com o objetivo de reduzir o tempo de processamento de execução deste e proporcionar uma solução mais eficiente na redução do consumo de energia.

A terceira é implementar o algoritmo GVerde em um ambiente para simulação de grades computacionais, permitindo assim validar o algoritmo em diversos ambientes, principalmente em grades computacionais com milhares de recursos. A simulação permite realizar diversos experimentos e combinar diferentes cenários utilizando muito menos tempo e manutenção da infraestrutura.

REFERÊNCIAS

ALMEIDA, J. P. V. *Impacto de plataformas de virtualização no consumo energético: um estudo comparativo entre Xen e KVM*. 67 f. Monografia (Trabalho de Conclusão do Curso de Engenharia de Computação) — Univercidade Federal do Rio Grande do Sul, Porto Alegre, 2012.

ARTHI, T.; HAMEAD, H. Energy aware cloud service provisioning approach for green computing environment. In: *Energy Efficient Technologies for Sustainability (ICEETS), 2013 International Conference on*. [S.l.: s.n.], 2013. p. 139–144.

BORRO, L. C. *Escalonamento em grades móveis: uma abordagem ciente do consumo de energia*. 73 f. Dissertação (Mestrado em Ciência - Ciência da Computação e Matemática Computacional) — Instituto de Ciências Matemáticas e de Computação - ICMC-USP, São Carlos, 2014.

CASANOVA, H. et al. Heuristics for scheduling parameter sweep applications in grid environments. *IEEE CS Pres - Proceedings of the 9th Heterogeneous Computing Workshop*, 2010.

CORMEN, T. H. et al. *Algoritmos - Teoria e Prática*. 2ª. ed. Rio de Janeiro: Elsevier, 2002.

COSTA, E. J. F. et al. Um avaliador automático de eficiência de algoritmos para ambiente educacionais de ensino de programação. *Computer on the Beach 2014 - Antigos Completos*, 2014.

COUTINHO, F.; CARVALHO, L. de; SANTANA, R. A workflow scheduling algorithm for optimizing energy-efficient grid resources usage. In: *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*. [S.l.: s.n.], 2011. p. 642–649.

FALAVINHA JUNIOR, J. N. et al. Avaliação de algoritmos de escalonamento em grids para diferenres configurações de ambiente. *SBC - V Workshop em Desempenho de Sistemas Computacionais e de Comunicação - Rio de janeiro*, p. 505 – 524, 2007.

FAVARIM, F.; FRAGA, J. S.; LUNG, L. C. Towards an opportunistic grid scheduling infrastructure based on tuple spaces. *J Internet Serv Appl*, p. 159–172, 2012.

FEO, T. A.; RESENDE, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, p. 8, 67–71, 1989.

FEOFILOFF, P. *Algoritmos em Linguagem C*. Rio de Janeiro: Elsevier Editora Ltda, 2009. ISBN 978-85-352-3249-3.

FERNáNDEZ, A.; MARCELINO, J.; MARQUES, P. Cloud computing. *Instituto Nacional de Propriedade Industrial*, 2011.

FOSTER, I. What is the grid? a three point checklist. 2002. Disponível em: <http://www.mcs.anl.gov/ itf/Articles/WhatIsTheGrid.pdf>. Acesso em: Janeiro de 2013.

FOSTER, I. et al. Cloud computing and grid computing 360-degree compared. In: . [S.l.: s.n.], 2008.

GOMES, B. T. P. et al. Scheduling strategies evalution for opportunistic grids. *Sysmposium on Computing Systems*, 2010.

GORACZKO, M. et al. *Joulemeter: Computational Energy Measurement and Optimization*. 2011. Disponível em: <http://research.microsoft.com/en-us/projects/joulemeter/default.aspx>. Acesso em: 15-03-2014.

GUADAGNIN, L. Avaliação de desempenho para algoritmos de escalonamento em grades computacionais. 2010.

GUDE, S. A suevey of green it: Metrics to express greennes in the it industry. Vrije Universiteit Amsterdam, 2010.

GUNARATNE, C.; CHRISTENSEN, K.; NORDMAN, B. Managing energy consumption costs in desktop pcs and lan switches with proxying, split tcp connections, and scaling of link speed. *Int. J. Network Manag.*, v.15, p. 297–310, 2005.

KRITHIKA, B.; KEERTHANA, N. Comparison of intel processor with amd processor with green computing. In: *Green Computing, Communication and Conservation of Energy (ICGCE), 2013 International Conference on*. [S.l.: s.n.], 2013. p. 737–742.

LI, Q.; ZHOU, M. The survey and future evolution of grren computing. *IEEE/ACM International Conference on Green Computing and Communications*, Chengdu - China, 2011.

MARTINEZ, F. H. V. Algoritmos e programação de computadores ii. In: *Faculdade de Computação - UFMS*. [S.l.: s.n.], 2011. f. 238.

MURUGESAN, S. Harnessing green it: Principles and practices. 2008.

OLIVEIRA, L. et al. Simulation of computational grids scheduling politics. In: *Information Systems and Technologies (CISTI), 2010 5th Iberian Conference on*. [S.l.: s.n.], 2010. p. 1–6.

OLIVEIRA, R. G.; CALVACANTE, S. V. Explorando o paralelismo de dados e de thread para atingir a eficiência energética do ponto de vista do desenvolvimento de software. Recife, 2010.

ORGERIE, A.-C.; ASSUNCAO, M. D. d.; LEFEVE, L. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 46, n. 4, p. 47:1–47:31, mar. 2014. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/2532637>.

PINTO, J. P. F. *Uso de Algoritmo Genético para Escalonamento de Tarefes em Grades Computacionais no Simulador GRIDSIM*. 52 f. Dissertação (Mestrado em Ciência da Computação) — Universidade do Estado do Rio Grande do Norte e Universidade Federal Rural do Semi-Árido, Mossoró - RN, 2013.

PRUHS, K. Green computing algorithmics. In: *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*. [S.l.: s.n.], 2011. p. 3–4. ISSN 0272-5428.

RANGE, M. C.; ABREU, N. M. M. d.; BOAVENTURA-NETTO, P. O. GRASP PARA O PQA: UM LIMITE DE ACEITAÇÃO PARA SOLUÇÕES INICIAIS. *Pesquisa Operacional*, scielo, v. 20, p. 45 – 58, 06 2000. ISSN 0101-7438.

RIVOIRE, S.; RANGANATHAN, P.; KOZYRAKIS, C. A comparison of higt-level full system power models. 2008.

ROCHA, L. A. *Abordagem H[ibidra para Alocaçao de Máquinas Virtuais em Nuvens Computacionais*. 99 f. Tese (Tese a Nível de Doutorado em Engenharia Elétrica) — Universidade Estatual de Campinas. Faculdade de Engenharia Elétrica e de Computação, Campinas-SP, 2013.

RUEST, N.; RUEST, D. *Virtualization, A Beginner's Guide*. Nova Yorque: McGraw Hill Publications, 2009.

SALGADO, G. T. *Estudo sobre Impacto Energético de Máquinas Virtuais em Sistemas Computacionais Físico*. Dissertação (Trabalho de Conclusão de Curso) — Universide Federal do Rio Grande do Sul - UFRGS, Porto Alegre, 2011.

SANTOS NETO, E. L. *Escalonamento de Aplicações que Processam Grandes Quantidades de Dados em Grids*. 83 f. Dissertação (Mestrado em Ciência da Computação) — Universide Federal de Campina Grande, Campina Grande, Paraíba, Brasil, 2014.

SILVA, A. H.; CARVALHO, R. A. M. Uma análise de um projeto de sustentabilidade focado em virtualização. São Caetano do Sul, 2011.

SILVA A., L. J. M. Cloud computing: Segurança e privacidade da informação na nuvem. *IESF - Instituto Superior de Estudos Financeiros e Fiscais*, 2012.

SILVA C., F. J. *Estratégias Baseadas em Políticas Verdes para Alocação de Recursos em Grids Computacionais*. 122 f. Tese (Tese de Doutorado em Engenharia de Sistema e Computação) — Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2012.

SILVA, D. P.; CIRNE, W.; BRASILEIRO, F. V. Trading cycles for information: Using replication to schedule bag-of-tasks aplications on computational grids. *Euro-Par 2003 Parallel Processing - Springer*, 2003.

SILVA J., C. *Avaliação Energética de Software: Uma Contribuição à Computação Verde*. 52 f. Monografia (Trabalho de Conclusão do Curso de Licenciatura em Ciência da Computação) — Universidade Federal da Paraíba, Rio Tinto-PB, 2012.

SILVA, L. P.; SANTOS, L. C. Uso da tecnologia de virtualização para aproveitamento de recursos de hardware. *Fasci-Tech - Periódico Eletrônico da FATEC*, São Caetano do Sul, v.1, 2010.

SILVA, M. R. P. et al. Ti verde - princípios e práticas sustentáveis para aplicação em universidades. 2010.

SILVA, R. M. da; MARTINO, R. C. *Integração de Aplicações de Grid ao Globus 4*. 92 f. — Universidade de Brasília, Brasília, 2007.

SOARES, F. B.; FAINA, L. F. Segurança em grandes computacionais. Uberlândia - MG, 2008.

SOROR, A. A. et al. Automatic virtual machine configuration for database workloads. *ACM Trans. Database Syst.*, p. 1–47, 2010.

STANOEVSKA-SLABEVA, K.; WOZNIAK, T.; RISTOL, S. *Grid and Cloud Computing - A Business Perspective on Technology and Applications*. 1ª. ed. New York: Springer, 2010.

TEODORO, S. *Algoritmos de Escalonamento para Grades Computacionais voltadas à Eficiência Energética*. 117 f. Dissertação (Mestrado em Ciência da Computação) — Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2013.

VILLARREAL, M. E. *Netpowercloudsim: Extensão do Cloudsim para Validação de um Modelo de Gerenciamento de Equipamentos de Rede Legados em Nuvem Verde*. 77 f. Monografia (Trabalho de Conclusão do Curso de Ciência da Computação) — Instituto Federal de Educação, Ciência e Tecnologia Catarinense - Campus Rio do Sul, Rio do Sul, 2013.

WANDERS, M. Data center verde: Como reduzir o impacto ambiental. p. 27–38, 2011.

WATANABE, E. et al. Algoritmos para economia de energia no escalonamento de workflows em nuvens computacionais. *32º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2014*, SBRC, Florianópolis, 2014.

WERNER, J. *Uma Abordagem para Alocação de Máquinas Virtuais em ambientes de Computação em Nuvem Verde*. 137 f. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Santa Catarina, Florianópolis, 2011.

YAMINI, R. Power management in cloud computing using green algorithm. *IEEE - International Conference On Advances In Engineering, Science And Management*, p. 6, 2012.

ZHANG, X.; GONG, L.; LI, J. Research on green computing evaluation system and method. *IEEE*, Beijing - China, 2011.

ANEXO A

CÓDIGO FONTE

```
package cliente;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Properties;
import org.apache.xmlrpc.XmlRpcException;
import org.apache.xmlrpc.client.XmlRpcClient;
import org.apache.xmlrpc.client.XmlRpcClientConfigImpl;

public class Cliente {

    public Cliente(int cod, String nome, String URL, String porta
     , int eMax, int eMin)
     throws MalformedURLException, XmlRpcException{

        File file = new File("propriedades.properties");
        Properties props = new Properties();
        FileInputStream fis = null;
        try {
            fis = new FileInputStream(file);
            //lê os dados que estão no arquivo
            props.load(fis);
            fis.close();
```

```
        }
        catch (IOException ex) {
            System.out.println(ex.getMessage());
            ex.printStackTrace();
        }

        String urlServidor = props.getProperty("urlServidor");
        System.out.println(urlServidor);

        XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
        config.setServerURL(new URL(urlServidor+":8080/xmlrpc"));
       // config.setServerURL(new URL("http://127.0.0.1:8080/xmlrpc"));
        XmlRpcClient cliente = new XmlRpcClient();
        cliente.setConfig(config);

        Object[] params = new Object[]{new Integer(cod), nome, URL+
         ":"+porta+"/xmlrpc",
         new Integer(eMax), new Integer(eMin)};
        boolean resultado =
          (Boolean) cliente.execute("GIS.adicionarRecurso"
         , params);
    }
}

package Nodo;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
```

```
import java.util.Properties;
import java.util.Scanner;
import org.apache.xmlrpc.XmlRpcException;
import servidor.Servidor;

public class Nodo {
    public static void main(String[] agrs) throws XmlRpcException,
      IOException{
        String nome = null;
        String URL = null;
        String porta = null;
        String eMax = null;
        String eMin = null;
        String cod = null;

        File file = new File("propriedades.properties");
        Properties props = new Properties();
        FileInputStream fis = null;
        try {
            fis = new FileInputStream(file);
            //lê os dados que estão no arquivo
            props.load(fis);
            fis.close();
        }
        catch (IOException ex) {
            System.out.println(ex.getMessage());
            ex.printStackTrace();
        }
```

```
nome = props.getProperty("nomeNodo");
URL = props.getProperty("urlNodo");
porta = props.getProperty("portaNodo");
eMax = props.getProperty("eMaxNodo");
eMin = props.getProperty("eMinNodo");
cod = props.getProperty("codNodo");

System.out.println("Cod: "+cod);
    System.out.println("Nodo: "+nome);
    System.out.println("URL: "+URL);
    System.out.println("Porta: "+porta);
    System.out.println("Consumo Max: "+eMax);
    System.out.println("Consumo Min: "+eMin);
    System.out.println("");

try{
    Servidor nodo = new Servidor(Integer.parseInt(cod), nome,
     URL, porta,
     Integer.parseInt(eMax),Integer.parseInt(eMin));
    //Servidor nodo =
     new Servidor(nome, URL, Integer.parseInt(eee));

    System.out.println("Cod: "+cod);
    System.out.println("Nodo: "+nome);
    System.out.println("URL: "+URL);
    System.out.println("Porta: "+porta);
    System.out.println("Consumo Max: "+eMax);
    System.out.println("Consumo Min: "+eMin);
    System.out.println("");
    System.out.println("Conectado...");
```

```
            System.out.println("");
        }catch(Exception e){
            System.out.println("Erro ao fazer
              a conexação com o servidor!");
            e.toString();
        }

    }
}

package servidor.servicos;

public class Servicos {

    public int selectionSort(int tamanho) {
        long tempoInicial = System.currentTimeMillis()/1000;

        System.out.println("Ordenar um
         vetor pelo método SelectionSort...");
        System.out.println("Tamanho do vetor: "+tamanho);

        int numeros[] = gerarVetorInteiro(tamanho);
        int index_min;
        int aux;

        imprimirVetor(numeros);

        for (int i = 0; i < numeros.length-1; i++) {
            index_min = i;
            for (int j = i + 1; j < numeros.length; j++) {
```

```
            if (numeros[j] < numeros[index_min]) {
                index_min = j;
            }
        }
        if (index_min != i) {
            aux = numeros[index_min];
            numeros[index_min] = numeros[i];
            numeros[i] = aux;
        }
    }

    int tempo =
     (int) ((System.currentTimeMillis()/1000) - tempoInicial);
    imprimirVetor(numeros);
    return tempo;
}

public int InsertionSort(int tamanho) {
    long tempoInicial = System.currentTimeMillis()/1000;

    System.out.println("Ordenar um
     vetor pelo método InsertionSort...");
    System.out.println("Tamanho do vetor: "+tamanho);

    int numeros[] = gerarVetorInteiro(tamanho);
    imprimirVetor(numeros);

    int eleito;

    for (int i = 1; i < numeros.length; i++) {
```

```
            eleito = numeros[i];
            int j;
            for (j = i-1; (j >= 0) && (numeros[j] > eleito); j--) {
                numeros[j+1] = numeros[j];
            }
            numeros[j+1] = eleito;
        }
        int tempo =
         (int) ((System.currentTimeMillis()/1000) - tempoInicial);
        imprimirVetor(numeros);
        return tempo;
    }

    private int[] gerarVetorInteiro(int tamanho){
        System.out.println("Gerando um
         vetor de tamanho: "+tamanho);
        int[] vetor = new int[tamanho];

        for(int i = 0; i < vetor.length; i++) {
            int rnd = (int) (1 + Math.random() * tamanho) ;
            vetor[i] = rnd;
        }
        return vetor;
    }

    private void imprimirVetor(int[] vetor){
        System.out.print("Vetor[ ");
        for(int i = 0; i < vetor.length; i++){
            System.out.print(vetor[i]+" ");
        }
```

```
        System.out.print("] ");
        System.out.println("");
    }
}

package servidor;

import ServidorPrincipal.GIS;
import cliente.Cliente;
import java.io.IOException;
import org.apache.xmlrpc.XmlRpcException;
import servidor.servicos.Calculadora;
import org.apache.xmlrpc.server.PropertyHandlerMapping;
import org.apache.xmlrpc.server.XmlRpcServer;
import org.apache.xmlrpc.server.XmlRpcServerConfigImpl;
import org.apache.xmlrpc.webserver.WebServer;
import servidor.servicos.Servicos;
import servidor.servicos.TempoDeEspera;

public class Servidor {
    public Servidor(int cod, String nome,
     String URL, String porta, int eMax,
     int eMin)
     throws XmlRpcException, IOException{
        System.out.println("Porta: "+porta);
        WebServer webServer =
         new WebServer(Integer.parseInt(porta));

        XmlRpcServer xmlRpcServer =
         webServer.getXmlRpcServer();
```

```
        PropertyHandlerMapping phm =
         new PropertyHandlerMapping();
        phm.addHandler("Calculadora", Calculadora.class);
        phm.addHandler("TempoDeEspera",
         TempoDeEspera.class);
        phm.addHandler("Servicos", Servicos.class);
        xmlRpcServer.setHandlerMapping(phm);

        XmlRpcServerConfigImpl serverConfig =
         (XmlRpcServerConfigImpl) xmlRpcServer.getConfig();
        serverConfig.setEnabledForExtensions(true);
        serverConfig.setContentLengthOptional(false);

        webServer.start();

        Cliente cliente =
         new Cliente(cod, nome, URL, porta, eMax, eMin);

        System.out.println("Nodo "+nome+" em Execução!");
    }

    public void desconectar(int porta){
        WebServer webServer = new WebServer(porta);
        webServer.shutdown();
    }
}

package ServidorPrincipal.Escalonadores;
```

```
import ServidorPrincipal.AnalisadorVerde;
import ServidorPrincipal.GIS;
import ServidorPrincipal.Recurso;
import ServidorPrincipal.ServidorPrincipal;
import ServidorPrincipal.Tarefa;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;

public class EscGrasp {
    public EscGrasp(){

    }

    public void body() throws InterruptedException{

        System.out.println("Algoritmo De Escalonamento
          GRASP em Execução...");
        System.out.println("Gerando
         Ordem de Execução...");
        System.out.println("");
        List<Executar> ordemDeExecucaoFinal =
         Grasp(50,GIS.listaTarefas.size());

        System.out.println("Ordem de Execução Gerada:");
        System.out.println("");
        Iterator it = ordemDeExecucaoFinal.iterator();
```

```
        while(it.hasNext()){
            Executar exec = (Executar) it.next();
        }
        System.out.println("");
        System.out.println("Executando...");
        System.out.println("");
        executar(ordemDeExecucaoFinal);
    }

    private List<Executar> Grasp(int p, int q){
        List<Recurso> listaR = GIS.getListaRecursos();
        List<Executar> solucao = new LinkedList<Executar>();
        List<Executar> melhorSolucao = new LinkedList<>();
        solucao = ordemDeExecucaoInicial(listaR);
        melhorSolucao = solucao;
        int cont = 0;
        boolean parar = true;

        while(parar){
            solucao = ordemDeExecucaoInicial(listaR);
            solucao = buscaLocal(solucao, q);
            if(calcularTempoDaOrdem(solucao) <
             calcularTempoDaOrdem(melhorSolucao)){
                melhorSolucao = solucao;
            }
            cont++;
            if(cont==p){
                parar = false;
            }
        }
```

```
        return melhorSolucao;
    }

    private List<Executar> buscaLocal(List<Executar>
      solucao, int q){
        List<Executar> solucaoTemp =
         new LinkedList<Executar>();
        int cont =0;

        Random randomE = new Random();

        solucaoTemp = solucao;
        for(int i = 0; i < q; i++){
            int e =
             randomE.nextInt(solucaoTemp.size()-1);
            Executar exec = solucaoTemp.get(e);
            int index = escolherRecursoPorMenorTempo(
             exec.getTarefa(), GIS.getListaRecursos());
            Recurso r = GIS.listaRecursos.get(index);
            exec.setRecurso(r.getCod());
            solucaoTemp.set(e, exec);

            if(calcularTempoDaOrdem(solucao)>
             calcularTempoDaOrdem(solucaoTemp)){
                solucao = solucaoTemp;
                cont++;
           }else{
                solucaoTemp = solucao;
           }
        }
```

```
        return solucao;
    }

     private int escolherRecursoPorMenorTempo(
      int tarefa, List<Recurso> listaRecursos) {
        int index = 0;
        Recurso r = listaRecursos.get(index);
        double tempo =
         GIS.obterTempoDeTarefaEmRecurso(tarefa, r.getCod());

        for(int i = 1; i < listaRecursos.size(); i++){
            Recurso recursoTemp = listaRecursos.get(i);
            double tempoTemp = GIS.obterTempoDeTarefaEmRecurso(
             tarefa, recursoTemp.getCod());
            if(tempo > tempoTemp){
                tempo = tempoTemp;
                index = i;
            }
        }

        return index;
    }

    private double calcularTempoDaOrdem(List<Executar> lista){
        double tempoTotal = 0;
        Hashtable<Integer, Double> tempoDosRecursos =
         new Hashtable<Integer, Double>();


        //iniciado as tabelas
```

```
        for(int i = 0; i < GIS.listaRecursos.size(); i++){
            tempoDosRecursos.put(GIS.listaRecursos.get(i).getCod(),
              0.0);
        }

        for(int i = 0; i < lista.size(); i++){
            int t = lista.get(i).getTarefa();
            int r = lista.get(i).getRecurso();

            Double tempoT = (Double)
             (GIS.obterTempoDeTarefaEmRecurso(t, r)/1000);
            tempoDosRecursos.put(r, tempoDosRecursos.get(r)+tempoT);
        }

        for(int i = 0; i < GIS.listaRecursos.size(); i++){
            tempoTotal = tempoTotal + tempoDosRecursos.get(
             GIS.listaRecursos.get(i).getCod());
        }
        return tempoTotal;
    }
}

package ServidorPrincipal.Escalonadores;

import ServidorPrincipal.AnalisadorVerde;
import ServidorPrincipal.Auxiliar.ComparadorTarefaEEE;
import ServidorPrincipal.Auxiliar.ComparadorTarefaEEEGrasp;
import ServidorPrincipal.GIS;
import ServidorPrincipal.Recurso;
import ServidorPrincipal.ServidorPrincipal;
```

```
import ServidorPrincipal.Tarefa;
import java.util.Collections;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;

public class EscGrasp1 {
    public EscGrasp1(){

    }

    public void body() throws InterruptedException{

        System.out.println("Algoritmo De Escalonamento GRASP em Execução...");
        System.out.println("Gerando Ordem de Execução...");
        System.out.println("");
        List<Executar> ordemDeExecucaoFinal = Grasp(50,GIS.listaTarefas.size());

        System.out.println("Ordem de Execução Gerada:");
        System.out.println("");
        Iterator it = ordemDeExecucaoFinal.iterator();

        while(it.hasNext()){
            Executar exec = (Executar) it.next();
        }
        System.out.println("");
        System.out.println("Executando...");
        System.out.println("");
```

```
        executar(ordemDeExecucaoFinal);
    }

    private List<Executar> Grasp(int p, int q){
        List<Recurso> listaR = GIS.getListaRecursos();
        List<Executar> solucao = new LinkedList<Executar>();
        List<Executar> melhorSolucao = new LinkedList<>();
        solucao = ordemDeExecucaoInicial(listaR);
        melhorSolucao = solucao;
        int cont = 0;
        boolean parar = true;

        while(parar){
            solucao = ordemDeExecucaoInicial(listaR);
            solucao = buscaLocal(solucao, q);
            if(calcularTempoDaOrdem(solucao) <
             calcularTempoDaOrdem(melhorSolucao)){
                melhorSolucao = solucao;
            }
            cont++;
            if(cont==p){
                parar = false;
            }
        }
        return melhorSolucao;
    }

    private List<Executar> buscaLocal(List<Executar> solucao, int q){
        List<Executar> solucaoTemp = new LinkedList<Executar>();
        int cont =0;
```

```
        Random randomE = new Random();

        solucaoTemp = solucao;

        for(int i = 0; i < q; i++){
            int e = randomE.nextInt(solucaoTemp.size()-1);
            Executar exec = solucaoTemp.get(e);
            int index = escolherRecursoPorEE(exec.getTarefa(),
             GIS.getListaRecursos());
            Recurso r = GIS.listaRecursos.get(index);
            exec.setRecurso(r.getCod());
            solucaoTemp.set(e, exec);

            if(calcularConsumoDeOrdem(solucao) >
             calcularConsumoDeOrdem(solucaoTemp)){
                solucao = solucaoTemp;
                cont++;
           }else{
                solucaoTemp = solucao;
           }
        }
        return solucao;
    }

    private double calcularConsumoDeOrdem(List<Executar> lista){
        double consumoTotal= 0.0;
        double consumoExecucao = 0.0;
        double consumoOcioso = 0.0;
```

```
Hashtable<Integer, Double> tempoDosRecursos =
 new Hashtable<Integer, Double>();
Hashtable<Integer, Double> consumoDosRecursos =
 new Hashtable<Integer, Double>();

//iniciado as tabelas
for(int i = 0; i < GIS.listaRecursos.size(); i++){
    tempoDosRecursos.put(GIS.listaRecursos.get(i).getCod(), 0.0);
    consumoDosRecursos.put(GIS.listaRecursos.get(i).getCod(), 0.0);
}

//pegar os dados de consum
for(int i = 0; i < lista.size(); i++){
    int t = lista.get(i).getTarefa();
    int r = lista.get(i).getRecurso();

    Double tempoT =
     (Double) (GIS.obterTempoDeTarefaEmRecurso(t, r)/1000);
    consumoDosRecursos.put(r, consumoDosRecursos.get(r)+
     (Double)(tempoT * GIS.getRecurso(r).getEMax()));
    tempoDosRecursos.put(r, tempoDosRecursos.get(r)+tempoT);
    consumoExecucao = consumoExecucao +
     (Double)(tempoT * GIS.getRecurso(r).getEMax());
}

 int codMaior = GIS.listaRecursos.get(0).getCod();
 Double tMaior = tempoDosRecursos.get(codMaior);

    //selecionar o maior e o menor
for(int i = 1; i < GIS.listaRecursos.size(); i++){
```

```
            int codTest = GIS.listaRecursos.get(i).getCod();
            Double cTest = tempoDosRecursos.get(codTest);
            if(tMaior < cTest){
                codMaior = codTest;
                tMaior = cTest;
            }
        }

        for(int i = 0; i < GIS.listaRecursos.size(); i++){
            Recurso rec = GIS.listaRecursos.get(i);
            double tempoOcioso = (Double) tMaior -
             tempoDosRecursos.get(rec.getCod());
               consumoOcioso = consumoOcioso +
                (Double) (tempoOcioso * rec.getEMin());
        }

        consumoTotal = consumoExecucao + consumoOcioso;
        return consumoTotal;
    }

    private double calcularTempoDaOrdem(List<Executar> lista){
        double tempoTotal = 0;
        Hashtable<Integer, Double> tempoDosRecursos =
         new Hashtable<Integer, Double>();


        //iniciado as tabelas
        for(int i = 0; i < GIS.listaRecursos.size(); i++){
            tempoDosRecursos.put(GIS.listaRecursos.get(i).getCod(), 0.0);
        }
```

```
        for(int i = 0; i < lista.size(); i++){
            int t = lista.get(i).getTarefa();
            int r = lista.get(i).getRecurso();

            Double tempoT = (Double) (
             GIS.obterTempoDeTarefaEmRecurso(t, r)/1000);
            tempoDosRecursos.put(r, tempoDosRecursos.get(r)+tempoT);
        }

        for(int i = 0; i < GIS.listaRecursos.size(); i++){
            tempoTotal = tempoTotal +
             tempoDosRecursos.get(GIS.listaRecursos.get(i).getCod());
        }
        return tempoTotal;
    }

    private List<Executar> ordemDeExecucaoInicial(
     List<Recurso> recursos){
        List<Executar> ordemInicial = new LinkedList<Executar>();

        int t = 0;

        Random gerarRecurso = new Random();

        ComparadorTarefaEEE comparador2 = new ComparadorTarefaEEE();
        Collections.sort(GIS.listaTarefas, comparador2);

        for(int i = 0; i < GIS.listaTarefas.size(); i++){
            t = gerarRecurso.nextInt(recursos.size());
```

```
            Executar exec = new Executar(GIS.listaTarefas.get(i).getCod(),
              recursos.get(t).getCod());
            ordemInicial.add(exec);
        }
        return ordemInicial;
    }
}

package ServidorPrincipal.Escalonadores;

import ServidorPrincipal.AnalisadorVerde;
import ServidorPrincipal.EnviarTarefa;
import ServidorPrincipal.GIS;
import ServidorPrincipal.ServidorPrincipal;
import java.net.MalformedURLException;
import java.util.Scanner;
import org.apache.xmlrpc.XmlRpcException;

public class Workqueue {

    public Workqueue(){

    }

    public void body() throws MalformedURLException,
      XmlRpcException, InterruptedException{
        int tamanhoListaRecursos = GIS.listaRecursos.size();
        int i = 0;

        ServidorPrincipal.tempoInicial = (System.currentTimeMillis())/1000;
```

```
boolean parada = true;
while(parada){
    Thread.sleep(300);
    if(GIS.listaRecursos.size()!=0){
        //Recurso recurso = GIS.listaRecursos.get(0);
        //GIS.listaRecursos.remove(0);
        if(GIS.listaTarefas.size()!=0){
            EnviarTarefa enviarTarefa = new EnviarTarefa();
            Thread threadEnviarTarefa = new Thread(enviarTarefa);
            threadEnviarTarefa.start();
        }else{
            System.err.println("Sem Tarefa disponível");
            while(parada){
                //parada = !();
                if(GIS.listaRecursos.size()==tamanhoListaRecursos)
                    parada = false;
                    Thread.sleep(500);
            }
        }
    }else{
       // System.err.println("Sem Recurso disponível");
    }
}
System.out.println("");
System.out.println("Simulação Completa...");
System.out.println("");

long tempoFinal = System.currentTimeMillis()/1000;

GIS.setTempoTotal(tempoFinal - ServidorPrincipal.tempoInicial);
```

```
            System.out.println("Tempo de Simulação:
             "+GIS.getTempoTotal()+" s");
            AnalisadorVerde.analizarConsumo();
            GIS.imprimirConsumo();
    }
}

package ServidorPrincipal;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.xmlrpc.XmlRpcException;
import org.apache.xmlrpc.client.XmlRpcClient;
import org.apache.xmlrpc.client.XmlRpcClientConfigImpl;

public class EnviarTarefa implements Runnable{

    public void run() {
        Recurso recurso = GIS.listaRecursos.get(0);
        GIS.listaRecursos.remove(0);
        Tarefa tarefa = GIS.listaTarefas.get(0);
        GIS.listaTarefas.remove(0);
        try {
            enviar(recurso, tarefa);
        } catch (MalformedURLException ex) {
            Logger.getLogger(EnviarTarefa.class.getName()).log(
             Level.SEVERE, null, ex);
```

```
        } catch (XmlRpcException ex) {
            Logger.getLogger(EnviarTarefa.class.getName()).log(
             Level.SEVERE, null, ex);
        }
    }

    private void enviar(Recurso recurso, Tarefa tarefa)
     throws MalformedURLException, XmlRpcException{

        System.out.println("Enviando tarefa "+tarefa.getId()+" "+
         tarefa.getNome()+" para o Recurso "+recurso.getNome());

        XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
        config.setServerURL(new URL(recurso.getUrl()));
        XmlRpcClient cliente = new XmlRpcClient();
        cliente.setConfig(config);

        Object[] params = tarefa.getParams();
        Integer result = (Integer) cliente.execute(tarefa.getNome(), params);
        System.err.println("Tempo de Execução da tarefa "+tarefa.getNome()
         +" pelo recurso "+recurso.getNome()+" foi "+result+" s");
        recurso.setTempoDeUso(recurso.getTempoDeUso()+result);
        GIS.listaRecursos.add(recurso);
    }
}

package ServidorPrincipal;

import java.util.ArrayList;
import java.util.Iterator;
```

```
import java.util.LinkedList;
import java.util.List;

public class GIS {
    public static List<Recurso> listaRecursos = new LinkedList<Recurso>();
    public static List<Tarefa> listaTarefas = new LinkedList<Tarefa>();
    public static List<Recurso> listaGR1 = new LinkedList<Recurso>();
    public static List<Recurso> listaGR2 = new LinkedList<Recurso>();
    public static List<Tarefa> listaGT1 = new LinkedList<Tarefa>();
    public static List<Tarefa> listaGT2 = new LinkedList<Tarefa>();
    public static long tempoTotal = 0;
    public static long consumoTotal = 0;
    public static int contRecursoUso = 0;
    public static int contRecursoG1 = 0;
    public static int contRecursoG2 = 0;

    public static void setContRecursoUso(int contRecursoUso){
        GIS.contRecursoUso = contRecursoUso;
    }

    public static int getContRecursoUso(){
        return contRecursoUso;
    }

    public static void setConsumoTotal(long consumoTotal){
        GIS.consumoTotal = consumoTotal;
    }

    public static long getConsumoTotal(){
        return GIS.consumoTotal;
```

```
}

public static void setTempoTotal(long tempoTotal){
    GIS.tempoTotal = tempoTotal;
}

public static long getTempoTotal(){
    return GIS.tempoTotal;
}

public static List<Recurso> getListaRecursos() {
    return listaRecursos;
}

public static List<Tarefa> getListaTarefas(){
    return listaTarefas;
}

public static boolean setListaRecursos(List<Recurso> listaRecursos) {
    try{
        GIS.listaRecursos = listaRecursos;
        return true;
    }catch(Exception e){
        e.printStackTrace();
        return false;
    }
}

 public boolean adicionarRecursoGIS(int cod, String nome, String URL
  , int eMax, int eMin){
```

```
        try{
            int id = listaRecursos.size();

            Recurso novoRecruso = new Recurso(id, cod, nome,
URL, eMax, eMin, 1);
            listaRecursos.add(novoRecruso);
            return true;
        }catch(Exception e){
            e.printStackTrace();
            return false;
        }
    }

    public boolean adicionarRecurso(int cod, String nome, String URL
     , int eMax, int eMin){
        try{
            adicionarRecursoGIS(cod, nome, URL, eMax, eMin);
            imprimirListaRecursos();
            return true;
        }catch(Exception e){
            e.printStackTrace();
            return false;
        }
    }

    public static Tarefa getTarefa(int cod){
        Tarefa t = null;
        Iterator it = listaTarefas.iterator();
        while(it.hasNext()){
            t = (Tarefa) it.next();
```

```
            if(t.getCod() == cod){
                return t;
            }
        }
        return t;
    }
}

package ServidorPrincipal;

public class Recurso {
    public int id;
    public String nome;
    public String url;
    public long eMax;
    public long eMin;
    public int status; //0 - Não disponível      1 - disponível
    public long consumoTotal;
    public long tempoDeUso;
    public int grupo;
    public int cod;
    public int numTarefa;

    public Recurso(int id, int cod, String nome, String url, long eMax,
      long eMin, int status) {
        this.id = id;
        this.nome = nome;
        this.url = url;
        this.eMax = eMax;
        this.eMin = eMin;
```

```
        this.status = status;
        this.consumoTotal = 0;
        this.tempoDeUso = 0;
        this.grupo = 0;
        this.cod = cod;
        this.numTarefa = 0;
    }
}

package ServidorPrincipal;

import ServidorPrincipal.Escalonadores.EscGrasp;
import ServidorPrincipal.Escalonadores.EscGrasp1;
import ServidorPrincipal.Escalonadores.EscGraspVerde;
import ServidorPrincipal.Escalonadores.EscMMV;
import ServidorPrincipal.Escalonadores.EscVerdeX;
import ServidorPrincipal.Escalonadores.MaxMinVerde;
import ServidorPrincipal.Escalonadores.Workqueue;
import java.net.MalformedURLException;
import java.util.Scanner;
import org.apache.xmlrpc.XmlRpcException;
import org.apache.xmlrpc.server.PropertyHandlerMapping;
import org.apache.xmlrpc.server.XmlRpcServer;
import org.apache.xmlrpc.server.XmlRpcServerConfigImpl;
import org.apache.xmlrpc.webserver.WebServer;

public class ServidorPrincipal {
    public static long tempoInicial = 0;

    private static final int port = 8080;
```

```
public static void main(String[] args) throws Exception
{
    GIS.criarListaDeTarefas();
    WebServer webServer = new WebServer(port);
    XmlRpcServer xmlRpcServer = webServer.getXmlRpcServer();

    PropertyHandlerMapping phm = new PropertyHandlerMapping();
    phm.addHandler("GIS", GIS.class);
    xmlRpcServer.setHandlerMapping(phm);

    XmlRpcServerConfigImpl serverConfig =
     (XmlRpcServerConfigImpl) xmlRpcServer.getConfig();
    serverConfig.setEnabledForExtensions(true);
    serverConfig.setContentLengthOptional(false);

    webServer.start();
    System.out.println("Grid Verde em Execução!");

    do{
        System.out.println("Iniciar Simulação?");
        String s = entrada.nextLine();
        if(s.equals("s")){
            System.out.println("Escolha qual o Algoritmo:");
            System.out.println("1 - Workqueue");
            System.out.println("2 - MMV");
            System.out.println("3 - GraspVerde");
            System.out.println("4 - EscVerdeX");
            System.out.println("5 - EscGrasp");
            System.out.println("6 - GVerde");
```

```
                s = entrada.nextLine();
                gerenciador(s);
            }
        }while(true);
    }
}

package ServidorPrincipal;

public class Tarefa {
    public String nome;
    public int id;
    public int peso;
    public Object[] params;
    public int cod;

    public Tarefa(int id, String nome, int peso, Object[] params, int cod){
        this.nome = nome;
        this.id = id;
        this.peso = peso;
        this.params = params;
        this.cod = cod;
    }
}
```