Escalonamento em grids computacionais: estudo de caso

Valéria Quadros dos Reis

Escalonamento em grids computacionais: estudo de caso

Valéria Quadros dos Reis

Orientador: Prof. Dr. Marcos José Santana

Monografia apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Julho de 2005

Sumário

Lista de Figuras

Lista de Tabelas

Agradecimentos

Agradeço a meus amados pais, Antonio e Rosângela, pelo apoio e incentivo que sempre me deram na vida. Sei que a luta não foi somente minha, então essa conquista também é de vocês.

Ao meu orientador, Professor Marcos José Santana, por ter acreditado em mim, ter entendido as minhas limitações e me aconselhado de maneira sábia quando tudo parecia dar errado.

À minha querida irmã Vanessa que torceu desde o começo e que entendeu a necessidade da nossa distância, assim como aos amigos e familiares que deixei em Campo Grande.

Às minhas grandes amigas Chris, Karen, Joselene e Hima Carla que fizeram a saudade de casa menor através de suas companhias e me mostraram a importância da amizade.

Ao meu querido amigo Eraldo Luís pelo companheirismo que demonstrou durante esse tempo.

Aos integrantes do meu grupo de pesquisa Thaís, Juliano, Michel, Bárbara, William, Caio, Gustavo, Júlio e Luís assim como aos Professores Regina Santana, Francisco Monaco e Sarita Bruschi.

Ao Professor Rodrigo Mello que me ajudou em uma fase decisiva do projeto e me mostrou o quanto é bom trabalhar no que se gosta.

Às muitas pessoas que conheci em São Carlos e que, de alguma forma, marcaram a minha vida. São muitos os momentos que lembrarei com saudades.

Agradeço também à CAPES pelo apoio financeiro concedido durante o desenvolvimento do projeto.

Por fim, e mais importante, agradeço a Deus por ter me dado forças para vencer esse desafio e, principalmente, por ter colocado pessoas tão especiais em meu caminho.

Resumo

Esta dissertação tem por objetivo apresentar a proposta de uma política de escalonamento para *grids* computacionais. Essa política, intitulada *Dynamic* Max-Min2x, é orientada ao escalonamento de aplicações cujas tarefas não realizam comunicação entre si e visa a redução do tempo de resposta dessas aplicações através da utilização de atribuição dinâmica de tarefas e replicação das mesmas. Experimentos, feitos através de simulação, mostram que o tempo médio de resposta de aplicações utilizando-se a *Dynamic* Max-Min2x é inferior ao de outras políticas da literatura. Análises dos resultados desses experimentos apontam que esse tempo tende a ser mais atrativo principalmente quando as tarefas necessitam de muito processamento e quando há grande variação de carga no sistema, características comuns em *grids* computacionais. Além disso, esta dissertação apresenta a implementação de um *framework* utilizando-se o Globus *Toolkit*, onde é possível a inserção de políticas de escalonamento para a submissão inteligente de tarefas em um *grid* computacional.

Abstract

This Master thesis proposes a new grid scheduling policy called Dynamic Max-Min2x. This policy focuses on applications in which tasks do not communicate among themsenves and targets a response time reduction of these applications through the use of dynamic task distribution and replication techniques. Experiments, done using simulations, have shown that the response time related to Dynamic Max-Min2x is smaller than others policies found in literature. Analysis of the results have demonstrated that this time tends to become more attractive when tasks do not need much processing power and when there is a great load variation in the system, characteristics frequently found in grids. Furthermore, this thesis presents the implementation of a framework using Globus Toolkit, which makes possible the new scheduling policies insertion to provide an intelligent submission tasks in a computational grid system.

.

Capítulo
1

Introdução

1.1 Contextualização

Apesar do recente surgimento da indústria da informática, em meados do século XX, ela sofreu um crescimento vertiginoso se comparado ao de outras indústrias como, por exemplo, a automobilística. Em pouco mais de 50 anos, foi possível migrar de máquinas compartilhadas por vários terminais para microcomputadores independentes, com grande capacidade computacional e interligados por redes de alta velocidade. Não diferente do crescimento do processamento dessas máquinas, a complexidade das aplicações também aumentou consideravelmente, forçando a busca por sistemas que processem as informações cada vez mais rapidamente (TANENBAUM, 2001).

Os primeiros computadores, conhecidos como *mainframes*, possuíam Sistemas Operacionais capazes de atender a diversos usuários simultaneamente, graças às técnicas de multiprogramação e compartilhamento de tempo que eram realizados. Apesar das aplicações que esses computadores executavam serem compostas, basicamente, de cálculos numéricos ou, simplesmente, editores de texto, eles custavam muito caro e, por esse motivo, havia poucas instituições com capital disponível para investir nesses equipamentos (SILBERSCHATZ *et al.*, 2001).

Com o avanço da eletrônica surgiram os circuitos VLSI (*Very Large Scale Integration*) que, permitindo agrupar milhões de transistores em uma única pastilha de silício, possibilitaram o desenvolvimento de uma nova classe de computadores, os microcomputadores. Essa classe possuía preços algumas ordens de magnitude menor que um computador de grande

porte e, assim, popularizou o uso dos computadores fora das grandes empresas.

O contínuo crescimento da complexidade de aplicações incentivou a prática do particionamento de programas para serem executados em máquinas paralelas, com o objetivo de diminuir o tempo de espera necessário para a obtenção de resultados. Porém, o uso de tais máquinas acarretava as desvantagens dos *mainframes*: custavam caro e, dependendo da aplicação, perdiam em desempenho para os microcomputadores.

Assim, tornou-se raro quem estivesse disposto a pagar o alto custo de uma máquina paralela, pois era possível atingir o mesmo desempenho com a união de um conjunto de microcomputadores. Porém, era preciso interligar essas máquinas para que houvesse troca de informação entre elas e compartilhamento de recursos. Não demorou muito para que esse objetivo fosse atingido com o uso de redes de computadores. Os sistemas centralizados, onde um *mainframe* era a unidade central de processamento, foram então, gradualmente, substituídos pelos Sistemas Distribuídos (TANENBAUM & MAARTEN, 2002).

Através de redes de computadores mais abrangentes e velozes e, principalmente, do surgimento e consolidação da rede mundial de computadores, a Internet, foi possível construir sistemas envolvendo máquinas geograficamente distribuídas, podendo estas pertencer a diferentes organizações e domínios. Nasceu assim, o conceito de *grids* computacionais, onde máquinas independentes, mesmo espalhadas fisicamente, são interconectadas para possibilitar o compartilhamento de recursos como, por exemplo, processamento, dados e serviços (FOSTER *et al.*, 2001; FOSTER & KESSELMAN, 1999).

Devido à sua natureza altamente distribuída, *grids* computacionais apresentam problemas tipicamente encontrados em Sistemas Distribuídos, tais como a sincronização de dados, limitações de comunicação e a segurança na utilização do sistema. Essas dificuldades não são difíceis de se entender, visto a dimensão que um *grid* pode alcançar, abrangendo diversos domínios, tipos de máquinas e de Sistemas Operacionais (STALLINGS, 2000; BERSTIS, 2002).

1.2 Motivação

Uma grande motivação para o uso de *grids* computacionais está no fato de poderem ser constituídos da união de recursos ociosos, muito freqüentemente encontrados em grandes instituições ou mesmo em domicílios. Essa característica resulta na formação de grandes computadores virtuais, onde é possível executar aplicações que exigem computação intensiva de dados. Por outro lado, já surgem empresas que vendem processamento a quem estiver in-

teressado em submeter seus trabalhos para serem executados em um *grid* computacional (SETIATHOME, 2005; ENTROPIA, 2005; PARABON, 2005; BUYYA *et al.*, 2001). A vantagem para um cliente em obter tais serviços é a obtenção de resultados mais rapidamente ou mesmo utilizar equipamentos sofisticados e de alto custo, que seriam inviáveis de se ter para uso próprio.

Tanto no primeiro quanto no segundo caso, é preciso haver um consenso entre o doador e o usuário dos recursos para que a utilização dos mesmos não prejudique nenhuma das partes envolvidas. Esse concenso é estabelecido através da adoção de políticas que decidem quando, quanto e por quem os recursos do sistema serão utilizados. A criação dessas políticas, contudo, não é uma tarefa trivial, pois deve satisfazer aos interesses de diversos usuários e dos proprietários dos recursos. A dificuldade está no fato de que os usuários desejam ser atendidos a qualquer momento e da melhor forma possível, enquanto que, em um sistema compartilhado, os recursos devem ser divididos de maneira a satisfazer a todos os usuários presentes sem que, contudo, o proprietário perca a autoridade sobre o seu recurso.

Por essas razões é que a pesquisa na área de escalonamento de processos em *grids* computacionais tem se tornado fundamental. São diversos os esforços para o desenvolvimento de políticas de escalonamento, ambientes de comercialização de recursos e escalonadores integrados com os serviços existentes em um sistema para computação em *grid* (MAHESWARAN *et al.*, 1999; SILVA, 2003; VADHIYAR & DONGARRA, 2002; BUYYA *et al.*, 2001; JAMES *et al.*, 2002).

Assim, este projeto de mestrado vem ao encontro desses trabalhos, propondo um sistema para submissão de tarefas embutido no conjunto de ferramentas do Globus, um dos mais difundidos *middlewares* para construção de *grids* computacionais existente. A intenção é que esse sistema possibilite a atribuição inteligente de tarefas à plataforma de acordo com regras de escalonamento determinadas pelo próprio usuário. Dessa forma, também é apresentada uma nova política de escalonamento, chamada *Dynamic* Max-Min2x, para a atribuição de tarefas aos recursos de um *grid* de maneira a diminuir o tempo total de espera pelos resultados de uma determinada aplicação. O objetivo é que este trabalho contribua na consolidação de *grids* computacionais como uma opção viável de plataforma para a execução de aplicações com grande demanda de processamento.

1.3 Estrutura da Monografia

Esta monografia está organizada em seis capítulos que descrevem a revisão bibliográfica sobre a área de escalonamento em *grids* computacionais, o desenvolvimento deste projeto, os experimentos e os resultados obtidos e as conclusões do trabalho. Neste primeiro capítulo é feita uma introdução ao assunto tratado neste documento, apresentando o contexto e a motivação deste trabalho.

O Capítulo 2, intitulado "Computação Paralela Distribuída", tem por objetivo esclarecer ao leitor os conceitos básicos sobre Computação Paralela e Sistemas Distribuídos. Esse capítulo é dirigido de forma a mostrar que essas duas definições têm convergido e, hoje, são utilizadas quase que indistintamente, formando o que se chama de Computação Paralela Distribuída, área onde os *grids* computacionais se integram como plataforma de execução.

No Capítulo 3 são expostos conceitos sobre *grids*, suas definições, características e similaridades com outros sistemas computacionais, os serviços necessários para o funcionamento dessa infra-estrutura, assim como uma introdução aos mais importantes sistemas de computação em *grid* existentes na atualidade.

O Capítulo 4 é dedicado à descrição em maior profundidade do Globus, um dos sistemas mais utilizados para a formação e utilização de *grids* computacionais. Nesse capítulo, são apresentados os principais serviços oferecidos pelo Globus, assim como os meios de implementação e utilização dos mesmos.

O Capítulo 5, por sua vez, insere noções sobre escalonamento de processos, classificações e taxonomias presentes nessa área de pesquisa. Nesse capítulo é feita a diferenciação entre políticas e algoritmos de escalonamento, destacando-se os resultados desejáveis de se atingir com a adoção de uma atribuição de tarefas adequada. Algumas políticas são apresentadas visando destacar as principais idéias exploradas em escalonamentos de processos em *grid*.

O desenvolvimento do projeto é detalhadamente descrito no Capítulo 6, onde há a proposta, estudo e análise de uma nova política de escalonamento, a *Dynamic* Max-Min2x, que visa otimizar o tempo total de espera para a coleta dos resultados de aplicações no *grid*. Esse capítulo também introduz detalhes da utilização dos serviços disponibilizados pelo Globus na implementação de um sistema flexível para a submissão de tarefas em sistemas de *grid*, onde é possível a utilização de diferentes políticas de escalonamento de acordo com as necessidades do usuário.

Por fim, o Capítulo 7 expõe as conclusões do trabalho apresentado através de análise

dos resultados obtidos, destacando as principais contribuições deste projeto assim como as propostas para trabalhos futuros.

Capítulo 2

Computação Paralela Distribuída

2.1 Considerações Iniciais

Desde o surgimento dos computadores, já existia a idéia de incluir paralelismo na execução de sistemas de computação. Isso se deu pelo desejo de alcançar tempos de resposta cada vez mais rápidos através da execução de instruções em paralelo, em aplicações que necessitam de grande quantidade de processamento (AMORIM *et al.*, 1988). Este capítulo tratará da área computacional originada pela busca do aumento do desempenho das aplicações, que tem se juntado à técnica de compartilhamento de recursos resultando no que se chama de Computação Paralela Distribuída. Durante o decorrer do capítulo, a taxonomia proposta por Flynn, em 1972, é introduzida para exemplificar os diversos tipos de arquiteturas de computadores existentes. Para finalizar, é dada a contextualização de *grids* computacionais como um sistema pertencente à Computação Paralela Distribuída, uma vez que, pode-se utilizar *grids* tanto para melhoria de desempenho quanto para compartilhamento de recursos.

2.2 Computação Paralela e Computação Distribuída

De acordo com Almasi (ALMASI & GOTTLIEB, 1994), Computação Paralela é entendida como "Uma grande coleção de elementos de processamento que podem se comunicar e cooperar entre si para resolver problemas de forma mais rápida que o método seqüencial". Essa definição data de 1989, porém permanece até os dias atuais pela sua grande abrangência na definição de Sistemas de Computação Paralela.

Em uma máquina tipicamente paralela, os processadores são interligados por barramentos de alta velocidade, compartilham a mesma memória e dispositivos de entrada e saída, e são compostos todos de um mesmo tipo de processador. A grande desvantagem de máquinas paralelas está concentrada em seu alto custo de aquisição, sendo que o investimento feito em um equipamento como esse pode ser indesejável devido à rapidez de processamento que máquinas seqüenciais têm alcançado. Entre as principais arquiteturas tipicamente paralelas, podem-se citar os SMPs (*Symmetric Multiprocessors*), as NUMA (*Nonuniform Memory Access*) e os processadores vetoriais (STALLINGS, 2000).

Quando um grupo de elementos de processamento, que se comunicam para trabalhar em conjunto, encontra-se distribuído em máquinas distintas, está caracterizado um tipo de Sistema de Computação chamado Sistemas Distribuídos. De acordo com Coulouris (COULOURIS *et al.*, 2001), Sistemas Distribuídos consistem em "sistemas nos quais os componentes localizados em uma rede de computadores se comunicam e coordenam suas ações somente através de troca de mensagens". Por essa definição é possível inferir que em um Sistema Distribuído há concorrência de processos e inexistência de um relógio global. Porém, tais características se apresentam de forma transparente para o usuário que utiliza o sistema sem saber dos meios que seu programa usufrui para ser executado.

De uma maneira geral, a Computação Paralela é feita para se atingir melhorias no desempenho, motivo pelo qual é comumente referenciada por Computação de Alto Desempenho. Sistemas Distribuídos, por sua vez, visam o compartilhamento de recursos, tanto lógicos quanto físicos, sendo que o trabalho realizado em um sistema desse tipo recebe o nome de Computação Distribuída. Esse objetivo da Computação Distribuída tem levado a um crescimento de sua prática, pois, nos dias atuais, são poucas as instituições capazes de adquirir equipamentos sofisticados para o seu uso dedicado, e mesmo quando o recurso a ser compartilhado trata-se de dados, a Computação Distribuída se faz presente através de projetos colaborativos, onde os resultados colhidos de experimentos são disponibilizados para diferentes equipes de pesquisa.

A grande vantagem dos Sistemas Distribuídos reside no seu atraente custo x benefício, pois podem ser formados até mesmo pela união de computadores pessoais. Tal característica, unida à busca por maior desempenho, tem levado à realização da Computação Paralela sobre Sistemas Distribuídos, sendo, por esse motivo, chamada de Computação Paralela Distribuída (AMORIM *et al.*, 1988; SOUZA, 2000).

Na Computação Paralela Distribuída, um conjunto de máquinas independentes é interligado para simular um computador paralelo, compondo o que se chama de máquina

paralela virtual. O fato desses sistemas serem formados por máquinas independentes que não compartilham a mesma memória, faz com que eles sejam chamados de sistemas fracamente acoplados, ao contrário das máquinas tipicamente paralelas ditas fortemente acopladas (STALLINGS, 2000).

Sistemas fracamente acoplados resultam em algumas dificuldades na programação. O problema da comunicação, por exemplo, é solucionado através da formação de um esquema em que toda vez que processo deseja enviar informações para outro, ele manda uma mensagem contendo essa informação ao destinatário. Assim, a troca de informações é fundamental em plataformas como essas e pode ser feita através dos Ambientes de Passagem de Mensagem, cujos representantes mais difundidos são o PVM (*Parallel Virtual Machine*) e o MPI (*Message Passing Interface*) (BUYYA, 1999).

Ambientes de Passagem de Mensagem são extensões de linguagens seqüenciais disponibilizadas através de bibliotecas de comunicação. Esses ambientes são responsáveis por permitir que aplicações paralelas possam ser construídas e executadas utilizando-se um sistema distribuído sendo que, para atingir esse objetivo, eles devem tratar de problemas tais como a heterogeneidade de arquiteturas e de seus formatos de dados.

2.3 A Classificação de Flynn

Flynn (FLYNN, 1972), ainda no começo da década de 70, apresentou uma classificação das arquiteturas de computadores de acordo com o fluxo de instruções e o fluxo de dados existentes em cada sistema. Apesar de existirem outras, a classificação de Flynn é amplamente utilizada até os dias atuais devido à grande abrangência que a taxonomia ainda possui (STALLINGS, 2000; AMORIM *et al.*, 1988).

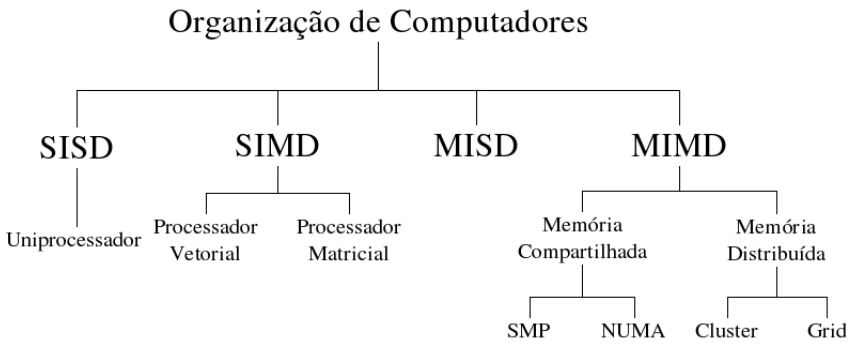

Figura 2.1: Extensão da Classificação das arquiteturas de processadores segundo Flynn.

A figura 2.1 ilustra uma extensão das quatro categorias de arquiteturas existentes,

segundo a Taxonomia de Flynn, juntamente com exemplos de cada categoria. As seguintes classes são formadoras dessa taxonomia:

- **Single Instruction Single Data stream (SISD)** - essa classe engloba os computadores seqüenciais onde existe uma única unidade de controle responsável por decodificar as instruções que serão realizadas sobre um conjunto de dados. A figura 2.2 ilustra o modelo das arquiteturas SISD.

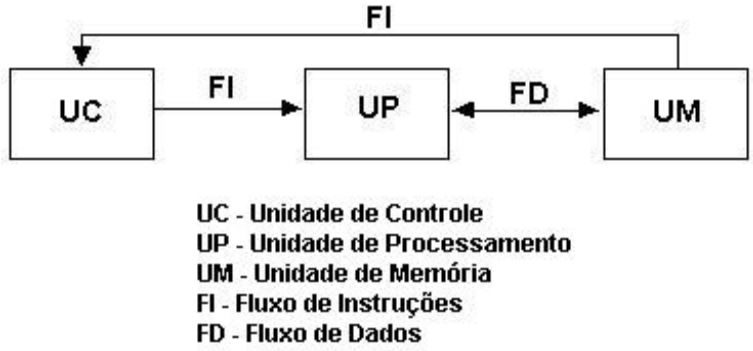

Figura 2.2: Modelo de arquitetura SISD.

- **Single Instruction Multiple Data stream (SIMD)** - uma grande representação da classe SIMD consiste nos computadores vetoriais e matriciais, onde vários processadores recebem as instruções de uma única unidade de controle e as executam sobre diferentes dados armazenados em suas respectivas áreas de memória assim como mostra a figura 2.3.

- **Multiple Instruction Single Data stream (MISD)** - apesar de alguns autores como, por exemplo, Almasi (ALMASI & GOTTLIEB, 1994) considerarem máquinas com técnica de *pipeline* representantes da classe MISD, ela não possui nenhuma implementação que se enquadre, de fato, nas suas características, onde um mesmo conjunto de dados é distribuído a um conjunto de processadores que realizam diferentes instruções sobre esses dados. Um esquema de como seria um computador MISD é exibida pela figura 2.4.

- **Multiple Instruction Multiple Data stream (MIMD)** - a classe MIMD é a representante das arquiteturas que realizam múltiplas instruções simultaneamente e, por isso, a que mais tem se destacado ao longo do tempo. Nela, um conjunto de processadores executa diferentes seqüências de instruções sobre diferentes conjuntos de dados. Como é possível ver na figura 2.1, a organização dos processadores MIMD divide-se nos

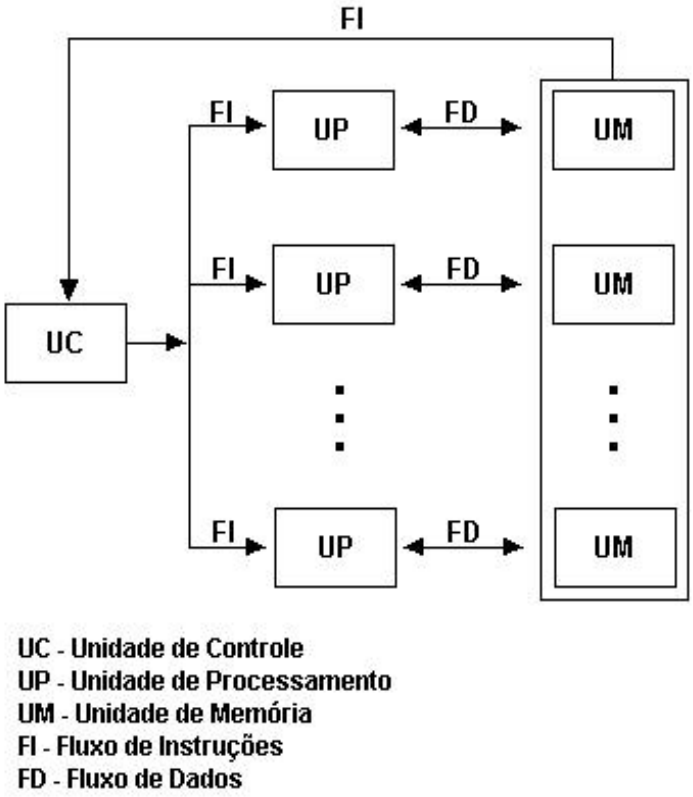

Figura 2.3: Modelo de arquitetura SIMD.

processadores de memória compartilhada, também chamados de fortemente acoplados e nos de memória distribuída, chamados de fracamente acoplados. Existe ainda uma outra nomenclatura para esses tipos de sistemas, sendo que o conjunto de processadores que compartilham a mesma memória é chamado de multiprocessador, tendo os SMPs e as NUMAs como representantes, e o conjunto de processadores independentes, que não compartilham memória, forma os chamados multicomputadores. Nesse último grupo estão os *clusters* e os *grids* computacionais. A figura 2.5 faz uma ilustração de uma organização de processadores MIMD.

2.4 Grids Computacionais na Computação Paralela Distribuída

Grids Computacionais constituem uma plataforma que se adequa à Computação Paralela Distribuída, pois são freqüentemente utilizados tanto para computação de alto desempenho quanto para o compartilhamento de recursos.

De fato, o conceito de *grid* também se adequa à definição de Sistemas Distribuídos

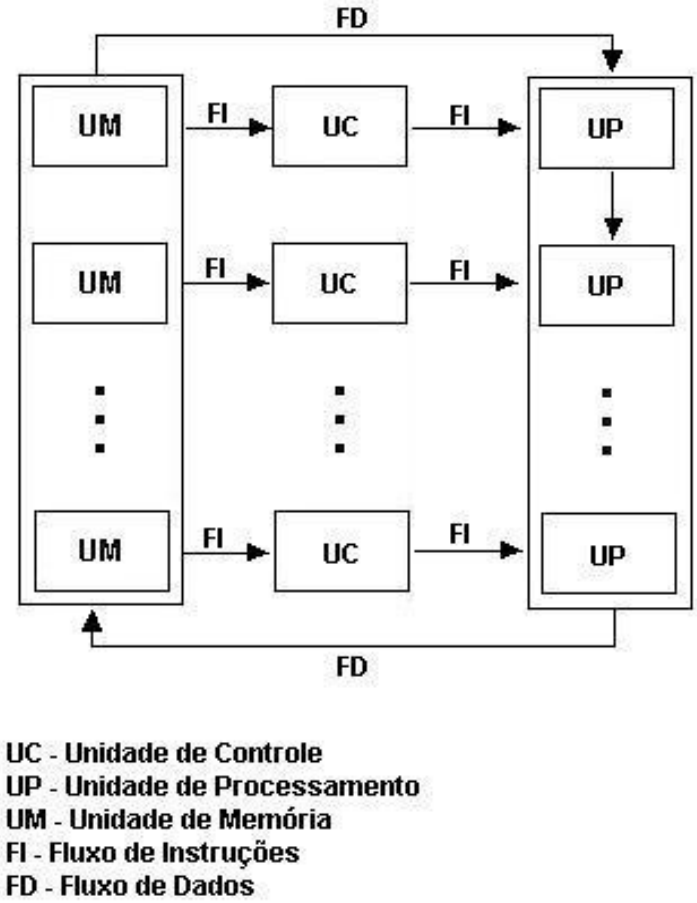

Figura 2.4: Modelo de arquitetura MISD.

de Tanenbaum que diz que "Um Sistema Distribuído é uma coleção de computadores independentes que parecem um único computador para o usuário do sistema" (TANENBAUM & MAARTEN, 2002). A transparência evidenciada nessa frase é uma característica chave do conceito de *grids*, ao passo que, nesses sistemas, recursos são unidos para a construção de uma grande máquina virtual utilizada pelo usuário final. Existem ainda, outros atributos nativos de Sistemas Distribuídos e que se aplicam a *grids* computacionais. É o caso da heterogeneidade, da escalabilidade, da confiabilidade e do desempenho (STALLINGS, 2000).

Um *grid*, freqüentemente, é formado por diversos recursos espalhados geograficamente e que, por esse motivo, costumam não ser homogêneos. Porém, isso não impede a união de tais componentes em um único sistema. A escalabilidade do *grid* também é garantida, pois a adição de um recurso no sistema não implica em custos desproporcionais para o aumento do número de recursos.

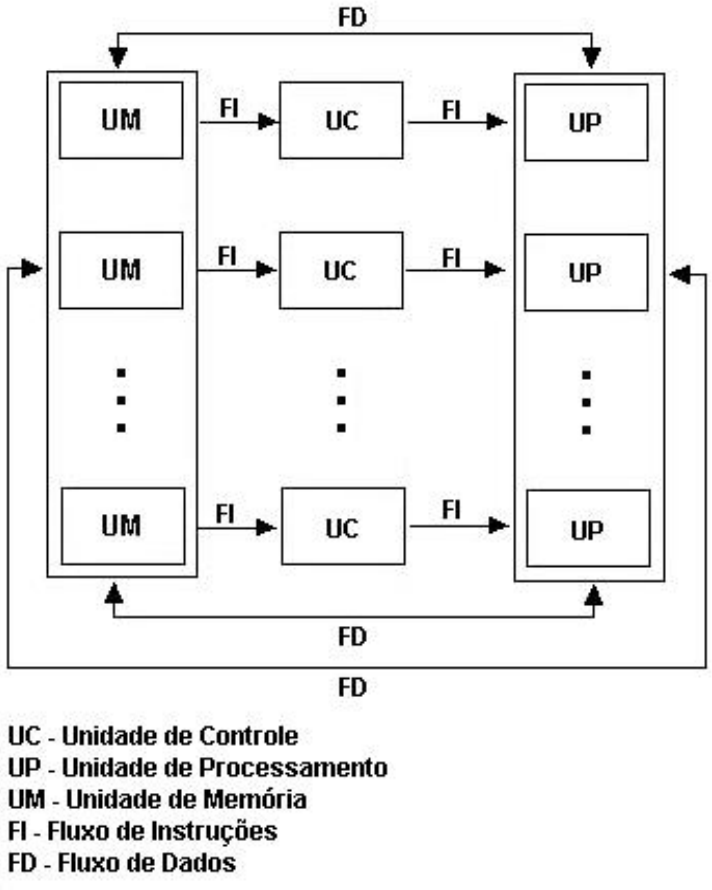

Figura 2.5: Modelo de arquitetura MIMD.

O fato de um *grid* ser formado por diversos recursos acarreta em maior confiabilidade do sistema, pois se um componente do sistema falhar, existem outros que podem sustituí-lo sem que haja uma grande perda de desempenho da aplicação sendo executada. O desempenho que, por sua vez, é alvo da Computação Paralela, é um dos objetivos chaves da construção de sistemas como o *grid* computacional, onde ganha-se tempo através do maior número de máquinas trabalhando em paralelo. Tal procedimento é, sem dúvida, um dos maiores atrativos desse tipo de plataforma.

2.5 Considerações Finais

A computação de dados utilizando-se máquinas tipicamente paralelas, apesar de representar um importante meio para a melhoria de desempenho de aplicações, possui custos bastante elevados, o que tem resultado na realização de Computação Paralela sobre Sistemas Distribuídos. Essa técnica, conhecida como Computação Paralela Distribuída, usufrui dos

custos mais acessíveis de um Sistema Distribuído, além de, muitas das vezes, beneficiar a execução de tarefas reduzindo o tempo de resposta das mesmas.

Dessa forma, as máquinas paralelas que, tipicamente, possuíam arquitetura SIMD, de acordo com a taxonomia de Flynn, passaram a ser, cada vez mais, substituídas por sistemas de computação distribuída com arquitetura MIMD, onde a técnica de *grids* computacionais surge como uma opção para a realização de Computação Paralela Distribuída.

O próximo capítulo será dedicado à apresentação formal do conceito de *grids* computacionais, da sua origem, da natureza dos serviços necessários em um sistema como esse, além da exemplificação de alguns projetos importantes envolvendo *grids* computacionais.

Capítulo 3

Grids Computacionais

3.1 Considerações Iniciais

A evolução de computadores e redes culminou com o aparecimento dos Sistemas Distribuídos. Esses sistemas apresentam custos de projeto, implementação e manutenção mais acessíveis do que sistemas baseados em máquinas paralelas. *Grids* vêm ao encontro dessa tendência ao unirem computadores em um sistema onde é possível a realização de Computação Paralela Distribuída. O pouco tempo de existência desse conceito, porém, faz com que o termo seja, constantemente, utilizado de forma errônea. Assim, neste capítulo, são apresentados a definição de *grids* computacionais, sua relação com algumas tecnologias existentes na computação, os serviços presentes nesse tipo de sistema, além de casos de sucesso envolvendo *grids* como plataforma de execução.

3.2 A Natureza de Grids Computacionais

Grids computacionais consistem em uma infra-estrutura formada por *hardware* e *software* que permitem o compartilhamento de recursos como, por exemplo, dados, capacidade de processamento e armazenamento. Esse compartilhamento se dá através de estruturas de interconexões, *softwares* de monitoramento e controle de informações. Além disso, *grids* devem ser seguros, consistentes, pervasivos e possuir um baixo custo de acesso (FOSTER, 2002; CIRNE, W., 2002; FOSTER & KESSELMAN, 1999).

Apesar do conceito de *grid* ter surgido na década de 90, ele é baseado em uma idéia

antiga: o compartilhamento de recursos em uma rede de computadores. No caso de *grids*, a rede, muitas vezes, trata-se da Internet. Dessa forma, os recursos pertencentes a um *grid* podem estar espalhados geograficamente, bastando que eles estejam conectados à rede mundial de computadores. Além disso, devido à natureza dos *grids*, os mesmos podem possuir recursos bastante heterogêneos, tanto física quanto logicamente, sendo que tais recursos comumente pertencem a diferentes organizações, sendo assim multi-institucionais. Essa última característica citada constitui um dos maiores problemas a serem resolvidos por um sistema de computação em *grid*, ou seja, como administrar recursos pertencentes a diversos domínios que, por sua vez, possuem suas próprias políticas de administração.

As entidades as quais compartilham recursos em um *grid* formam organizações virtuais ou simplemente VO´s como são conhecidas na área. O perfil mais comum dessas entidades são organizações de pesquisa, universidades e provedores de recursos, sejam eles de armazenamento, aplicação ou processamento (FOSTER *et al.*, 2001).

Várias são as motivações para a construção de *grids* computacionais. Uma delas é o fato de que apesar da capacidade de processamento dos computadores seguir a Lei de Moore e dobrar a cada 18 meses, ainda existem aplicações que necessitam de muitas máquinas trabalhando em paralelo, às vezes por diversos dias, para que elas sejam finalizadas. Muitas dessas aplicações são executadas em máquinas maciçamente paralelas, porém, a tendência é que esse procedimento seja cada vez mais raro devido ao elevado custo para aquisição de um equipamento como esse. Uma alternativa é o compartilhamento de supercomputadores entre diversas entidades, porém, o que se vê é uma clara migração dessas aplicações para ambientes paralelos virtuais que oferecem capacidade computacional a um custo mais acessível (FOSTER & KESSELMAN, 1999; GOLDCHLEGER *et al.*, 2002).

A grande potência de processamento que computadores pessoais têm obtido representa um outro incentivo para o uso de *grids*, uma vez que, essas máquinas têm hoje a capacidade computacional semelhante à de um supercomputador há 10 anos atrás. Além disso, é raro encontrar dentro de um ambiente comercial, residencial ou até mesmo acadêmico quem ocupe tal processamento (BERSTIS, 2002; FOSTER & KESSELMAN, 1999). A idéia é que outras pessoas possam utilizar a ociosidade de tais computadores, juntando recursos suficientes para a execução de aplicações com necessidades especiais.

3.3 Funcionalidades de um Grid

De acordo com a funcionalidade que oferecem, *grids* podem ser classificados em três categorias: de processamento, de dados e de serviços, assim como ilustra a figura 3.1 (BUYYA *et al.*, 2002).

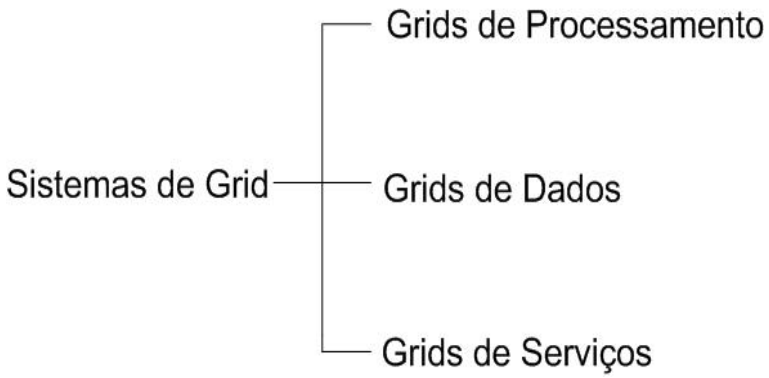

Figura 3.1: Tipos de *grid* de acordo com a sua funcionalidade.

3.3.1 Grids de Processamento

Nesse primeiro grupo de *grids* computacionais estão incluídas as aplicações que necessitam de grande capacidade computacional e que, por sua vez, não pode ser satisfeita com o uso de somente uma máquina. Dependendo de como essa capacidade é utilizada, esse tipo de *grid* pode ser empregado para Computação Distribuída ou para Computação Intensiva de Dados.

Aplicações de Computação Distribuída utilizam um agregado de recursos para solucionar problemas inviáveis de serem resolvidos por um recurso único devido ao tempo consumido para a obtenção de resultados. Exemplos de usos de Computação Distribuída são comumente encontrados em programas para simulações, estudo de parâmetros e otimização combinatória (GLOBUS, 2005).

Na Computação Intensiva de Dados o objetivo é extrair e sintetizar informações a respeito de uma grande quantidade de dados, sendo necessária a utilização de diversos recursos em paralelo. Programas para renderização de imagem e reconhecimento de padrões são exemplos de aplicações desta área.

3.3.2 Grids de Dados

A classe dos *grids* de dados é formada por uma infra-estrutura especializada em prover gerenciamento no armazenamento e acesso de dados. Um exemplo de aplicação que pode ser usada em um *grid* de dados consiste em um banco de dados altamente distribuído. Nesse caso, o papel do *grid* está em juntar as informações espalhadas e sintetizá-las para o uso de um cliente como se as mesmas estivessem em sua máquina local.

3.3.3 Grids de Serviço

O último grupo é chamado de *grid de serviço* devido à grande generalidade dos serviços que podem ser prestados por ele. Esse grupo se divide em três classes:

- **Colaborativa** - Na classe colaborativa, um conjunto de usuários compartilham um recurso em comum através de ambientes virtuais compartilhados, sendo que esse recurso pode ser um instrumento científico ou mesmo resultados obtidos de pesquisas. As aplicações usadas nessa classe possibilitam a comunicação em tempo real de seus integrantes e, dessa forma, indivíduos espalhados fisicamente podem trabalhar juntos em um mesmo projeto sem maiores dificuldades (GLOBUS, 2005);

- **Multimídia** - A classe de serviço multimídia oferece uma infra-estrutura para aplicações que fazem uso de diversas mídias em um mesmo intervalo de tempo, garantindo qualidade de serviço ao longo das muitas máquinas pertencentes ao *grid*;

- **Sob-demanda** - A classe sob-demanda, como o próprio nome sugere, presta serviços capazes de, dinamicamente, alocar mais recursos de acordo com a necessidade da aplicação. Na Computação Sob-demanda, além de processamento, pode-se compartilhar também *softwares*, dados e instrumentos científicos tais como sensores e telescópios. Nesse caso, os recursos do *grid* são compartilhados, por um tempo limitado, apenas quando os recursos locais são insuficientes, pois visam não somente um bom desempenho, mas também um bom custo.

É importante notar que a implementação de um *grid* que possua os três tipos de serviços citados é muito difícil de ser realizada, devido às características que cada classe de *grid* possui. Dessa forma, é comum a formação de *grids* especializados objetivando a satisfação do cliente com o serviço oferecido. Neste trabalho considerou-se, especialmente, o caso de *grids* de processamento, uma vez que, a política de escalonamento proposta se

relaciona à melhoria do desempenho de aplicações distribuídas com grande quantidade de processamento.

3.4 Infra-estruturas de Grid

Muitos são os investimentos na construção de infra-estruturas de *grids* computacionais. Algumas delas têm recebido mais destaque como é o caso do TeraGrid e do projeto FigthAidsAtHome da Companhia Entropia (TERAGRID, 2005; ENTROPIA, 2005).

O projeto TeraGrid foi, inicialmente, elaborado no ano de 2001 com recursos de quatro instituições norte-americanas, sendo elas: o Centro Nacional de Aplicações de Super Computação, localizado na Universidade de Illinois, o Centro de Super Computação de San Diego, localizado na Universidade da Califórnia, o Laboratório Nacional da Universidade de Chicago em Argonne e o Centro para Pesquisa de Computação Avançada no Instituto de Tecnologia da Califórnia. O objetivo principal do TeraGrid consiste na construção de uma grande infra-estrutura computacional distribuída voltada para a pesquisa científica. Através do uso dos serviços disponibilizados é possível que uma aplicação submetida para execução nessa plataforma usufrua de mais de 20 teraflops de capacidade computacional integrados por uma rede de aproximadamente 40 gigabits por segundo.

Diferente do paradigma adotado pelo projeto TeraGrid, onde os recursos agregados correspondem, geralmente, a máquinas de grande capacidade computacional, crescem os interesses pela formação de plataformas utilizando-se recursos ociosos de computadores pessoais. Talvez o maior exemplo desse tipo de *grid* seja o projeto pioneiro SetiAtHome o qual utiliza computadores ligados à rede mundial para procurar por existência extraterrestre (SETI - *Search for Extraterrestrial Intelligence*). Nele, qualquer computador pode ser um colaborador doando capacidade ociosa de processamento para os cálculos dos sinais captados por um radio-telescópio. Em 2000, o número de processadores participantes do projeto era de 1.6 milhões espalhados por 224 países, atingindo a potência de cerca de 10 teraflops (SETIATHOME, 2005).

Inspirados no sucesso alcançado pelo SetiAtHome, outros projetos semelhantes surgiram, onde os objetivos a se alcançar com os processamentos geralmente representam um interesse relacionado com pesquisa médica (PARABON, 2005; ENTROPIA, 2005; UNITED DEVICES, 2005). Um grande representante dessa classe de aplicações é o projeto Fight-AIDS@Home o qual acelera as pesquisas na busca de drogas mais eficazes no combate à AIDS. Até o momento o projeto alcançou a marca de 60 mil máquinas donatárias distribuí-

das por 20 países. Esses recursos unidos representam um computador com 14 terabytes de memória e 1335 terabytes de armazenamento em disco, onde o pico desse super computador consiste em 31 trilhões de operações matemáticas por segundo.

3.5 Aplicações de um Grid

A intensidade de comunicação entre as tarefas de uma aplicação costuma determinar o tipo de plataforma na qual ela deve ser executada, sendo que, quanto maior o grau de comunicação, maior deve ser o acoplamento entre as máquinas da plataforma. Em um *grid*, os recursos, tipicamente, são interligados por grandes redes de computadores, onde a latência de comunicação é muito alta, principalmente se a rede trata-se da Internet. Dessa forma, as aplicações submetidas ao *grid* devem evitar trocar informações entre suas tarefas, e serem o mais independentes possível umas das outras. Por esses motivos, aplicações desse tipo recebem o nome de *Bag-of-Tasks*, sendo comum encontrá-las nas áreas de Computação Distribuída e Computação Intensiva de Dados (FOSTER & KESSELMAN, 1999; SILVA, 2003; CASANOVA *et al.*, 2000).

3.6 Serviços Básicos de um Grid

Devido à sua natureza multi-institucional e às suas características como alta heterogeneidade, grande escala e grande dispersão geográfica, os *grids* necessitam de serviços básicos para o seu funcionamento como, por exemplo, segurança de informações, autenticação na plataforma, gerenciamento de dados e recursos, e comunicação (FOSTER *et al.*, 2001; FERREIRA *et al.*, 2003a; FOSTER & KESSELMAN, 1999; JACOB *et al.*, 2003).

Em um *grid* existem diversas instituições com interesse em fazer uso dos recursos computacionais. Dessa forma, é necessário que haja componentes responsáveis por gerenciar os recursos do sistema, localizando o que está sendo subutilizado ou sobrecarregado, ajudando no escalonamento de aplicações ou consertando falhas do *grid* automaticamente. Um serviço de autenticação de usuários do *grid* também é necessário para que haja um controle de quem utiliza os recursos do *grid*. Essa autenticação é, tipicamente, feita com o uso de senhas criptografadas (BERSTIS, 2002).

A autenticação faz parte não só do gerenciamento de recursos como também é um serviço básico de segurança por ajudar no banimento do acesso de não-membros aos recursos do *grid*. É preciso evitar que um usuário mal intencionado possa executar um código mali-

cioso que cause danos no sistema ou roube informações do mesmo. Para que isso seja feito, um bom serviço de segurança deve também estar presente em um *grid* computacional.

A monitoração do estado e da utilização das máquinas pode ser feita não só para auxiliar num justo escalonamento dos recursos, mas também para calcular o quanto cada cliente está utilizando da estrutura como um todo. Esse serviço torna-se bastante interessante quando há o comércio de recursos como, por exemplo, capacidade computacional. Em recursos não dedicados como no caso de sistemas que buscam por máquinas ociosas, a monitoração do uso de tais recursos tem papel fundamental, pois deve-se garantir ao dono da máquina que a execução de suas aplicações não será prejudicada, sendo que os processos do *grid* sendo executados localmente terão, de alguma forma, sua prioridade baixada ou até mesmo serão paralisados quando o usuário local voltar a utilizar o seu instrumento de trabalho.

Um serviço especial para tratar a transferência de dados entre as máquinas deve ser implementado, além de um serviço para controle e submissão de tarefas. Esses serviços são encarregados de transferir os dados necessários para a execução de um programa e observar se houve algum erro durante essa etapa.

É visível que a comunicação é uma característica chave também em sistemas de *grid*, pois um processo em uma máquina deve ser capaz de enviar informações para outra máquina localizada em um ponto distante da origem. Essas informações podem ser sobre resultados obtidos em processamento, dados necessários para uma computação, ou estatísticas sobre o estado de recursos do sistema. O papel da comunicação em *grids* computacionais é muito importante, pois não poderia haver uma junção de recursos espalhados geograficamente se não houvesse um canal de comunicação ligando os mesmos.

Finalmente, a presença de um escalonador pode melhorar muito o desempenho das aplicações e o uso dos recursos em um *grid*. Sistemas onde a carga de trabalho é bastante variável requerem o uso de políticas de escalonamento próprias que lidem com a grande dinamicidade presente na plataforma.

Um pacote de *software* que implemente algumas das funções citadas é chamado de sistema para computação em *grid* e, freqüentemente, é implementado como um *middleware*. Exemplos são o Globus, o Legion, o Condor e o OurGrid (FERREIRA *et al.*, 2003b; GLOBUS, 2005; LEGION, 2005; OURGRID, 2005; CONDOR, 2005a).

3.7 Sistemas de Computação em Grid

Desde o surgimento dos primeiros *grids*, alguns projetos foram propostos com o objetivo de desenvolver um pacote de serviços que fosse o mais completo possível para ser utilizado em tais plataformas. Alguns deles foram criados dentro de universidades como é o caso do Globus, outros como, por exemplo, o Condor foram propostos por uma iniciativa privada.

3.7.1 Globus

O Globus desenvolvido no Laboratório Nacional Argonne e na Universidade do Sul da Califórnia, é considerado um padrão entre os sistemas de computação para *grid*, possuindo uma grande variedade de documentação. Ele é formado por diversos serviços independentes tais como o gerenciamento de recursos, o gerenciamento de dados e a segurança. (GLOBUS, 2005; FERREIRA *et al.*, 2003b,a).

No Globus, os serviços globais são formados pela junção de serviços locais, pois, muitas vezes, é inviável existir uma única máquina responsável por manter consistentes os estados de todas as outras pertencentes ao sistema. Nesse caso, percebe-se o gargalo que existiria caso tal máquina fosse configurada para responder sozinha por um serviço.

Os serviços locais de um sistema Globus devem manter uma interface o mais simples possível, além de manter um padrão visando ser compatível com as mais diferentes arquiteturas.

3.7.2 Legion

O Legion é um sistema desenvolvido na Universidade de Virgínia nos Estados Unidos. Ele é orientado a objetos e, por esse motivo, todas as entidades do sistema, tais como capacidade de processamento e memória, são representadas como objetos (LEGION, 2005). Algumas das características do projeto são: autonomia para os domínios, arquitetura escalável e ambiente de computação homogêneo. Uma grande desvantagem do Legion é o fato de que seus serviços são altamente integrados, não possibilitando, dessa maneira, o uso de apenas parte deles.

3.7.3 OurGrid

O OurGrid é um projeto desenvolvido na Universidade Federal de Campina Grande (UFCG) em parceria com a Hewlett-Packard (HP), o qual visa o desenvolvimento de soluções que possam popularizar o uso de *grids* computacionais que, mesmo apesar dos avanços, ainda está muito longe da realidade de muitos usuários. Para isso, ele possibilita a submissão de aplicações *Bag-of-Tasks* em redes *peer-to-peer* de forma segura desde que o usuário tenha permissão de acesso às máquinas escolhidas (OURGRID, 2005).

3.7.4 Condor

Sendo o mais antigo sistema de computação em *grid* dentre os citados, Condor é utilizado, principalmente, para sistemas de *grid* que fazem uso de recursos ociosos da rede (CONDOR, 2005a). Por essa razão, as aplicações mais apropriadas para serem submetidas a esse sistema são as fracamente acopladas, que necessitam de pouca comunicação, pois assim essa aplicação pode ser dividida em muitas tarefas que são submetidas a diversos processadores diferentes. Dessa forma, costuma-se dizer que o Condor tem como objetivo a alta vazão, isto é, alto *throughput* em vez de alto desempenho.

3.8 Grids e outras Técnicas Computacionais

A idéia de formar *grids* computacinais que compartilhassem recursos através de uma rede não surgiu como uma grande novidade para o mundo. Isso porque como a maioria das técnicas computacionais existentes, *grids* surgem do emprego de serviços já consagrados como é o caso da *World Wide Web* e de *clusters* (FOSTER *et al.*, 2001).

Muitas são as divergências no sentido de classificar um *cluster* como um tipo de *grid* computacional. Os *clusters* surgiram nos anos 80 e consistem no agrupamento de máquinas que executam uma aplicação como, por exemplo, o PVM (*Parallel Virtual Machine*) ou MPI (*Message Passing Interface*) responsável pela comunicação entre processos (BUYYA, 1999). O termo *grid* surgiu por volta de 1995 mesmo tendo o Condor realizando computação altamente distribuída ainda em 1985. Apesar de também consistirem na união de máquinas com o objetivo de aumentar a oferta de recursos, quando se fala em *grid*, geralmente refere-se a um conjunto de máquinas espalhadas geograficamente e que pertençam a mais de uma instituição (FOSTER & KESSELMAN, 1999). Dessas duas últimas características citadas surgem outras como maior heterogeneidade e compartilhamento de recursos, além da inex-

istência de um controle centralizado no sistema. O que se discute é se a falta de algumas dessas características acarreta a não-classificação de um sistema como *grid* computacional. O que se pode afirmar é que os *grids* devido à sua natureza necessitam de serviços especiais para simular um computador virtual (CIRNE, W., 2002). É o caso do serviço de gerenciamento de recursos, o qual submete e monitora a execução de aplicações no *grid*, do serviço de informação que coleta dados sobre a utilização dos recursos do sistema, do serviço de gerenciamento de dados o qual transfere arquivos de uma máquina para outra e do serviço de segurança necessário, por exemplo, para autenticar de forma única os usuários membros do *grid* através das diversas instituições fornecedoras de recursos.

Os protocolos utilizados nos serviços *Web* também facilitam a disseminação dos *grids* em escala mundial. Protocolos tais como o HTTP (*HyperText Transfer Protocol*), o TCP/IP (*Transport Control Protocol/Internet Protocol*) e linguagens como HTML (*HyperText Markup Language*) se tornaram padrões e, dessa forma, contribuíram para a ótima aceitação da Internet para a comunicação. Da mesma forma, esperam-se que padrões construídos com base nesses protocolos tornem os *grids* computacionais comuns não só para a comunicação em si, mas também para a utilização e compartilhamento de recursos disponíveis.

Muito se especula sobre o aparecimento de empresas que vendem processamento e recursos através da formação de *grids* computacionais, mas esta é uma ação que já vem sendo tomada pelos servidores de aplicação que, por uma quantia estipulada com o cliente, cedem direito para execução de aplicações instaladas nos servidores ou ainda espaço para armazenamento de páginas *Web* em seus computadores. Tais serviços tornaram-se comuns nos últimos anos e visam, principalmente, pessoas e empresas que não têm condições de manter um servidor, 24 horas por dia conectado à rede, ou não têm condições de adquirir uma máquina ou aplicação capaz de atender de maneira satisfatória às requisições recebidas.

Finalmente, a computação *Peer-to-Peer* implementa a troca de arquivos de uma maneira muito específica que vai além do modelo cliente-servidor tradicional e chega bem próximo ao modelo exigido pelos *grids* os quais necessitam ultrapassar limites institucionais. Essa técnica, porém, é muito específica, pois não utiliza protocolos que busquem uma padronização para que haja uma interoperabilidade entre as instituições, característica comum em um mercado onde os participantes estão sempre buscando monopólio. Uma outra preocupação é a facilidade que se tem para se obter acesso e compartilhar arquivos em um sistema *peer-to-peer*. Tal característica não é desejável em um *grid* computacional onde a segurança dos recursos deve ser zelada.

3.9 Mitos sobre Grids

Diversos são os mitos existentes sobre o uso de *grids* e suas aplicações. De certa maneira, os responsáveis por essas falsas afirmações são os próprios criadores e estudiosos desse sistema, pois os mesmos expõem o conceito como algo novo e inovador, ao passo que, *grids* representam uma nova etapa de ambientes paralelos virtuais e que não substituirão por completo os supercomputadores, não necessitarão de novos Sistemas Operacionais e nem tampouco resolverão os problemas de processamento de todo tipo de aplicação. A seguir são comentados alguns conceitos errôneos sobre *grids* computacionais (FOSTER *et al.*, 2001):

- ***Grids* são fontes inesgotáveis de recursos** - *grids* estão sob um controle para compartilhamento de recursos e, dessa forma, não há como utilizar uma quantidade demasiada e irrestrita dos mesmos, pois existem outros membros do sistema que, de igual maneira, desejam usufruir dessas fontes. Políticas de controle de acesso aos dispositivos e para contabilização de uso devem ser empregados rigidamente, assim como protocolos coletivos para fornecimento de recursos e informações de custo que ajudem no compartilhamento desses recursos.

- **Com a difusão dos *grids*, os supercomputadores não serão mais necessários** - apesar de *grids* formarem supercomputadores virtuais muito potentes, ainda existem aplicações fortemente acopladas que só podem ser executadas em máquinas paralelas de fato. Essas aplicações, geralmente, exigem baixa latência e realizam muita comunicação e, por isso, não podem desperdiçar tempo com transferência de mensagens de sincronização ou dados pela rede; em algumas vezes, é até difícil de paralelizar tais aplicações para serem executadas em um *grid* computacional. A ajuda que *grids* computacionais podem trazer para essa classe de aplicações é prover acesso a supercomputadores através do compartilhamento de recursos. Nesse caso, várias instituições podem fazer uso dos mesmos equipamentos, diminuindo assim os custos que cada uma teria para comprar seu próprio supercomputador.

- **É preciso um novo modelo de programação para *grids*** - de fato, *grids* possuem características próprias não encontradas em ambientes seqüenciais ou mesmo paralelos. É o caso dos recursos pertencerem a instituições diferentes e a disponibilidade de recursos variar no tempo. Porém, se uma camada de *software* que abstraia e encapsule tais característica for construída, então o modelo de programação será praticamente o mesmo.

- **O uso de *grids* requer um Sistema Operacional Distribuído** - em um *grid* não é necessário haver um Sistema Operacional Distribuído executando em cada máquina, até porque isso representaria uma dificuldade para integração de recursos ao *grid*, pois muitos membros não estariam dispostos a instalar um novo sistema em seus computadores. Em vez disso, *middlewares* são instalados e fazem o papel de tal Sistema Operacional, como, por exemplo, transparência de localização de recursos e segurança no *grid*.

3.10 Limitações de um Grid

Apesar dos grandes benefícios que um *grid* computacional traz a algumas aplicações, ele possui diversas limitações, principalmente relativo ao tipo de programa que pode ser executado nele. É importante frisar que nem toda aplicação pode ser paralelizada para ser executada em um *grid*; algumas requerem a reescrita de algoritmos e, conseqüentemente, da aplicação, o que, aliado à freqüente preocupação com o desempenho de tais programas, pode representar um grande despendimento de esforço para conseguir um tempo de resposta satisfatório (BERSTIS, 2002).

Uma outra limitação do *grid* consiste nas conseqüências que um membro pode sofrer ao destinar seus recursos a essa estrutura. A integração de recursos a um *grid* pode afetar de maneira drástica o desempenho e a confiabilidade de uma organização quando não há um gerenciamento de recursos e medidas de segurança com políticas justas e consistentes no *grid*. Sendo assim, um usuário não disponibilizará seus recursos se não se sentir seguro e tiver certeza de que suas aplicações não serão prejudicadas para que outras pertencentes ao *grid* possam ser executadas.

3.11 Considerações Finais

O desenvolvimento de técnicas computacionais já consagradas como, por exemplo, as redes de computadores e a formação de computadores paralelos virtuais, fez com que a idéia de interligar computadores localizados no mundo inteiro para simulação de uma grande máquina se tornasse realidade através dos *grids* computacionais. Para a sua implementação, porém, é necessário que esses sistemas possuam serviços especializados em prover transparência ao usuário, tarefa realizada pelos Sistemas de Computação em *Grid*, onde um dos principais representantes consiste no chamado *Globus Toolkit*. Dessa forma, o próximo

capítulo descreverá os principais serviços presentes no Globus, os mecanismos utilizados na implementação de tais serviços e como as funcionalidades dos mesmos podem ser utilizadas na construção de novas aplicações.

Capítulo 4

O Conjunto de Ferramentas Globus

4.1 Considerações Iniciais

Devido à grande expectativa gerada em torno da consolidação de *grids* computacionais, muitos sistemas para criação e utilização dessas plataformas têm sido propostos. Porém, a falta de padronização tem retardado o crescimento e a consolidação dos mesmos, sendo que o Globus é um dos projetos que tem obtido grande aceitação pelos administradores de *grids*. Por esse motivo, este capítulo é dedicado à descrição, formas de acesso e mecanismos de funcionamento dos serviços oferecidos pelo Globus *Toolkit*, mais especificamente, da segurança, do gerenciamento de dados e recursos, dos sistemas de informação e dos ambientes para o desenvolvimento de aplicações.

4.2 O Globus Toolkit

O Globus é um conjunto de ferramentas de código aberto que possibilita o compartilhamento de recursos de forma segura sem comprometer a autonomia local, mesmo quando é necessário ultrapassar limites institucionais e geográficos (GLOBUS, 2005).

Desde a sua primeira versão, em 1998, até a quarta e mais recente, o Globus tem conquistado o apoio de importantes projetos e tende a se tornar um padrão, de fato, entre os tantos sistemas para computação em *grid* propostos até o momento [1]. Isso porque o Globus abstrai a heterogeneidade existente no *grid*, seja ela de *hardware* ou de *software*,

[1] Neste documento, a versão do Globus Toolkit tida como referência para estudo e documentação consiste na versão 3.2.1

por meio de troca de informações através da Internet e, consequentemente, com a utilização de protocolos já consagrados, motivo pelo qual as últimas versões do Globus apresentam serviços desenvolvidos com base em *Web services* (FERREIRA *et al.*, 2004; SOTOMAYOR, 2003).

4.2.1 Web e Grid Services

Basicamente, *Web services* compõem um conjunto de recursos da computação distribuída que permite a criação de aplicações baseadas no modelo cliente-servidor. Eles possuem técnicas para descobrir componentes de *software* a serem acessados, métodos para acessar esses componentes e métodos para a identificação de provedores de serviço. *Web services* são independentes de plataforma e linguagem e, além disso, utilizam protocolos conhecidos como é o caso, por exemplo, do HTTP (*HyperText Transfer Protocol*). Entre outros padrões utilizados pelos *Web services* pode-se destacar o SOAP (*Simple Object Access Protocol*) e o WSDL (*Web Services Description Language*) (FERREIRA *et al.*, 2003b; W3C, 2005).

Web services, porém, não são transientes e nem guardam estados entre uma chamada e outra (*stateful services*), condições necessárias em um *grid* computacional. É nesse ponto que entram os *Grid services* como uma extensão baseada em *Web Services* que, de acordo com convenções, determina como é feita a comunicação entre clientes e serviços do *grid* (SOTOMAYOR, 2003).

Para estabelecer padrões para o desenvolvimento, emprego e implementação de tecnologias e implementações para *grid* foi criado o *Global Grid Forum* (GGF) que culminou nas especificações da OGSI (Open Grid Services Infrastructure) e da OGSA (Open Grid Service Architecture) as quais estabelecem a natureza dos serviços que respondem às mensagens dos protocolos presentes no *grid*. O objetivo é que as especificações das interfaces dos serviços sejam completamente separadas de suas implementações (GGF, 2005).

4.2.2 OGSI

A OGSI (*Open Grid Service Infrastructure*) define como construir um *Grid service*, focando nos meios de criação, gerenciamento e troca de informações entre eles (TUECKE *et al.*, 2003; FERREIRA *et al.*, 2003b).

Alguns *Grid services* são considerados fundamentais ao funcionamento de um *grid* computacional, sendo que mecanismos para nomeação, serviço de dados, notificação e ciclo

de vida devem estar presentes nesses serviços.

O mecanismo de nomeação é responsável por atribuir um nome único a cada instância de serviços e permitir o descobrimento de operações através desses nomes.

No serviço de dados existem operações para configurar os valores dos serviços de dados, notificar a mudança de algum desses valores e associar informações com alguma instância de serviço de dados.

Na notificação encontram-se interfaces responsáveis pela entrega de notas sobre mudanças nos serviços e, por fim, o mecanismo de ciclo de vida é importante em um *grid*, pois permite a destruição de uma instância de serviço que por algum motivo, de falha na execução, por exemplo, deve ser feita. É esse mecanismo de ciclo de vida que faz com que cada serviço instanciado receba um tempo de vida no momento de sua criação.

É importante entender que a OGSI é responsável somente pela especificação dos mecanismos necessários para o bom funcionamento dos *Grid services*, sendo que a forma como tais mecanismos serão implementados é responsabilidade da OGSA.

4.2.3 OGSA

Para que o compartilhamento de recursos entre diversas instituições seja possível é necessário que elas interajam de uma forma conhecida e consistente para todas. Sendo assim, o *Global Grid Forum*, especificou a chamada *Open Grid Service Architecture* (OGSA) para que essa interoperabilidade entre membros do *grid* fosse solucionada. O termo *architecture* refere-se a um conjunto bem definido de interfaces nas quais sistemas podem ser construídos, enquanto que o termo *open* é utilizado para expressar a extensibilidade da arquitetura. Por interpretar todo e qualquer recurso do *grid* como um serviço, a OGSA é dita ser orientada-a-serviço (FERREIRA *et al.*, 2004, 2003b).

Os principais objetivos da OGSA são (UNGER & HAYNOS, 2003):

- **Gerenciar recursos residentes em diferentes plataformas** - essa é uma preocupação encontrada desde o surgimento do conceito de *grids* computacionais e que deve ser tratada com o desenvolvimento da OGSA;
- **Proporcionar qualidade de serviço (QoS)** - o nível dos serviços prestados pelo *grid* deve ser de alta qualidade, sendo importante que os meios para atingir essa qualidade sejam feitos de forma mais transparente possível para os usuários;
- **Dispor de soluções para o reparo do sistema** - a extensão que um *grid* pode

possuir, o número de recursos e diferentes configurações que eles podem tomar geram proporções gigantescas, além de diferentes estados e falhas desses recursos. Assim, é desejável que soluções inteligentes de auto-regulagem e correção de falhas existam em um sistema de *grid*;

- **Definir interfaces públicas e abertas** - a criação de interfaces abertas torna a interoperabilidade entre recursos mais fácil, pois a disponibilidade universal das mesmas facilita a padronização de arquiteturas como a OGSA;

- **Explorar técnicas existentes e estabilizadas** - mais uma vez a tentativa de padronização da arquitetura é evidenciada. O uso de técnicas consagradas facilita a aceitação da arquitetura, pois não força o uso e desenvolvimento de novos protocolos e interfaces. A construção da OGSA, por exemplo, é feita com base nos *Web services*.

Durante a especificação dos *Grid services*, duas foram as funcionalidades adicionadas aos *Web services*. Uma está relacionada à natureza transiente do *grid*, onde instâncias de serviços são criadas e destruídas dinamicamente e, assim, esses serviços necessitam de interfaces que gerenciem sua criação, destruição e tempo de vida. A outra reserva-se ao estado de um serviço, que pode ser um atributo ou dado associado a ele como acontece com objetos no paradigma da programação orientação a objetos (SOTOMAYOR, 2003).

O Globus foi a primeira implementação a seguir as especificações da OGSA (GLOBUS, 2005).

4.3 Serviços Presentes no Globus

O Globus inclui serviços e bibliotecas para a segurança, gerenciamento de recursos e dados, detecção de falhas, portabilidade e coleta de informações (GLOBUS, 2005). Esse conjunto de funcionalidades pode ser utilizado de forma independente ou como um todo no desenvolvimento de aplicações. A seguir, cada uma dessas funcionalidades serão comentados com o objetivo de esclarecer os mecanismos de funcionamento presentes no conjunto de ferramentas.

4.3.1 Segurança

A segurança em *grids* computacionais é dificultada pela necessidade de se estabelecer um grande número de relações de confiança entre domínios administrativos. No Globus, ela

é provida pelo *Grid Security Infrastructure*(GSI) o qual é baseado nos conceitos de infra-estrutura de chaves públicas e certificados, além de fazer uso do protocolo *Security Socket Layer*(SSL) para a autenticação (GLOBUS, 2005; FERREIRA *et al.*, 2003a).

Todo usuário e serviço do Globus possui duas chaves para a criptografia de dados, uma disponibilizada aos outros usuários e serviços do *grid* e outra que deve permanecer seguramente armazenada, as quais são chamadas chaves públicas e privadas, respectivamente. Essas chaves são utilizadas no momento da autenticação e estabelecimento de um canal seguro para comunicação.

Para contornar o problema da autenticidade de um usuário, máquina ou serviço, são utilizados certificados que mapeiam as suas identidades às suas chaves públicas. A criação desses certificados é realizada por Autoridades Certificadoras as quais recebem as chaves públicas dos usuários e serviços e verificam se tais chaves pertencem àquelas entidades.

Caso um usuário desejar utilizar vários recursos, cada um necessitando de uma autenticação mútua, então é necessária a geração de muitos certificados e, por conseqüência, várias entradas da senha para acesso à chave privada do usuário. Para proporcionar transparência ao usuário, o GSI provê a capacidade de delegação através da criação de *proxies*. Tais *proxies* consistem em novos certificados contendo a identidade do usuário e atestados por eles próprios em vez de uma autoridade certificadora.

Quando *proxies* são utilizados, a autenticação ocorre conforme ilustrado na figura 4.1. Nela, a parte remota recebe dois certificados: um do *proxy* com a assinatura do usuário e outro do próprio usuário com assinatura da Autoridade Certificadora. Durante a autenticação, a chave pública do usuário, obtida a partir de seu certificado, é utilizada para validar a assinatura contida no certificado do *proxy*. A chave pública da Autoridade Certificadora, por sua vez, é utilizada para validar a assinatura contida no certificado do usuário.

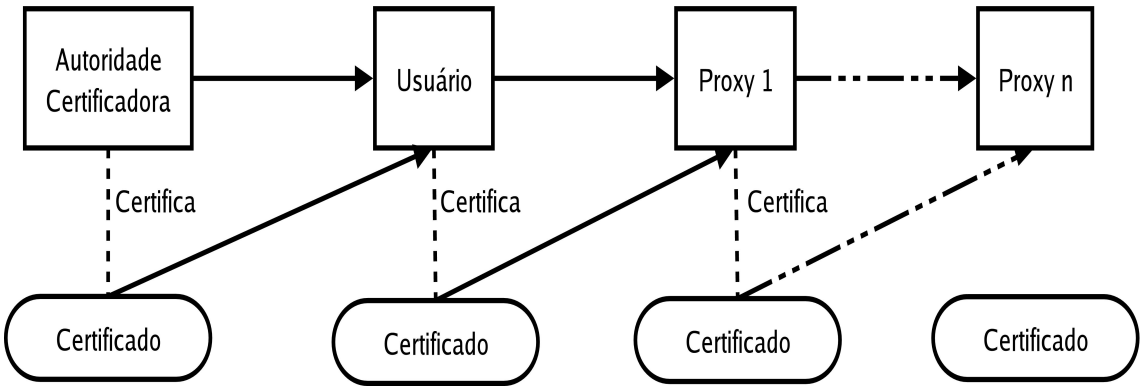

Figura 4.1: Mecanismo de delegação para autenticação única no *grid*.

Como medida de segurança, certificados de *proxies* possuem curtos tempos de vida. Dessa forma, se um desses certificados estiver em posse de pessoas erradas, os danos causados por ela podem ser limitados.

No caso de usuários que utilizam diferentes computadores para acessar serviços do *grid*, o Globus disponibiliza um sistema, chamado MyProxy, para a construção de servidores que centralizam o armazenamento de credenciais (identificações, chaves privadas) dos usuários para posterior geração de certificados. Essa medida é importante pois evita o depósito de dados do usuário em máquinas que não sejam confiáveis, conseqüentemente, aumentando a segurança do *grid* (LASZEWSKI *et al.*, 2004).

4.3.2 Gerenciamento de Dados

O esforço do Globus em relação ao gerenciamento de dados consiste na identificação, prototipação e avaliação de tecnologias que permitam a execução de aplicações que requerem acesso, algumas vezes altamente distribuído, a uma quantidade muito grande de dados da ordem de tera ou petabytes. Dessa forma, dois serviços muito importantes da área de gerenciamento de dados e presentes no conjunto de ferramentas do Globus são a transferência de dados e o gerenciamento de réplicas (GLOBUS, 2005; LASZEWSKI *et al.*, 2004; JACOB *et al.*, 2003).

Transferência de Dados

A característica do *grid* de ser multi-institucional, com diferentes formas de armazenamento, sistemas de segurança e sistemas operacionais, impossibilita o uso de diferentes tipos de protocolos para a comunicação entre aplicações. Tentando padronizar os protocolos utilizados, o Globus provê dois métodos para a transferência de dados: o GridFTP e o GASS, sendo que ambos utilizam o GSI para a autenticação segura dos dados.

Baseado no protocolo FTP (*File Transfer Protocol*), o GridFTP estende as funcionalidades de segurança, confiança e desempenho do FTP, escolhido como base para o GridFTP principalmente por ser o mais utilizado para a tranferência de dados na Internet e por oferecer as melhores vantagens para as necessidades de um *grid*.

Algumas características presentes no GridFTP são:

- Segurança pode ser baseada em GSI e Kerberos;
- Controle de transferência terceirizada a qual permite que um usuário ou aplicação em

A, possa monitorar e controlar uma transferência de dados entre B e C;

- Transferência paralela, segmentada e parcial de dados. Na transferência paralela, a movimentação de dados é melhorada através da abertura de múltiplos canais de comunicação entre a(s) fonte(s) e o destino. Na transferência segmentada, os dados são particionados e enviados para múltiplos servidores. Por fim, na trasferência parcial, apenas parte de arquivos são transferidos;

- Suporte para tranferências confiáveis de dados e possibilidade de reinício de transferência através do uso de *checkpoints*.

O GridFTP é disponibilizado como um serviço padrão em servidores utilizando o Globus *Toolkit*.

O GASS (*Globus Access Secondary Storage*) consiste em um mecanismo para leitura e escrita de arquivos remotos através do uso do protocolo HTTP (*HyperText Transfer Protocol*). Sendo assim, os arquivos a serem transferidos são acessados através de URLs (*Uniform Resource Locators*).

O GASS é utilizado, principalmente, pelo serviço de gerenciamento de recursos do Globus para a transferência de executáveis e arquivos necessários para a execução de uma aplicação. Da mesma forma, as saídas geradas também podem ser automaticamente transferidas sem que o usuário precise fazê-la manualmente.

Ao contrário do GridFTP, o GASS não provê meios para a disponibilização de arquivos para múltiplos usuários, apesar de poder utilizar um mesmo canal para o recebimento de arquivos de diferentes servidores. Assim, um usuário que deseje utilizar os serviços do GASS deve inicializar o seu próprio servidor.

Uma vantagem do GASS é a possibilidade de armazenar arquivos freqüentemente acessados, em *cache*. Essa característica é especialmente importante para a submissão de programas SPMD (*Single Program Multiple Data*), pois evita que o mesmo executável seja lido diversas vezes em disco, tipicamente uma mídia de baixa velocidade.

Um outro serviço presente no Globus é o *Reliable File Transfer Service* (RFT), que provê interfaces para a monitoração de transferências realizadas por terceiros.

Gerenciamento de Réplicas

Réplicas são capazes de reduzir a latência de acesso, a localização, a escalabilidade e o desempenho de aplicações distribuídas que utilizam grande quantidade de dados. O

Globus possui um serviço, chamado *Replica Location Service* (RLS) que provê as seguintes funcionalidades:

- Criação parcial ou total de novas réplicas;
- Registro de novas cópias em um Catálogo de Réplicas;
- Permissão para usuários e aplicações realizarem consultas nos catálogos para encontrar as localizações de cópias de determinados arquivos.

No Globus, cada dado possui um nome lógico que o identifica globalmente no *grid*, sendo que é papel do RLS realizar o mapeamento desses nomes lógicos para os nomes físicos reais, isto é, a localização física dos dados tratados.

4.3.3 Serviço de Informação

O alto desempenho, muitas vezes buscado em um *grid*, requer a seleção dos recursos mais aptos no momento da submissão da tarefa a ser executada. Para isso, o Globus disponibiliza Serviços de Informações, através dos quais é possível consultar o estado atualizado e preciso dos recursos presentes no *grid* assim como os serviços em funcionamento (JACOB *et al.*, 2003).

O Serviço de Informação funciona como um *Web service* que possui os estados de outros *Web services* presentes no *grid*. É papel do desenvolvedor de serviços especificar quais dados devem ser publicados e podem ser consultados por outros serviços. Além disso, também é possível a um cliente o pedido de acompanhamento dos estados de recursos assim como a notificação no momento de alguma mudança na informação (GLOBUS, 2005; SOTOMAYOR, 2003).

4.3.4 Gerenciamento de Recursos

O gerenciamento de recursos em *grids* computacionais é necessário para que seja possível a utilização de máquinas remotas como se fossem locais. Ele é responsável pelas requisições e alocações de recursos por aplicações, assim como pelo monitoramento da execução dessas aplicações. No Globus, o gerenciamento de recursos é realizado pelo *Grid Resource Allocation and Management* (GRAM) (GLOBUS, 2005; LASZEWSKI *et al.*, 2004).

O GRAM facilita o uso de recursos remotos através de uma interface para submissão e controle de tarefas no *grid*. Essa interface foi projetada para ser simples e flexível, pois

provê apenas um protocolo e uma única API (*Application Protocol Interface*) para utilização dos recursos, além de permitir a adoção de diferentes escalonadores. Por essa característica de simplicidade, a interface do GRAM pode ser representada pelo centro da ampulheta ilustrada na figura 4.2, sendo que na parte superior dela encontram-se as aplicações e os meta-escalonadores, e na parte inferior, os escalonadores locais. Por esse esquema, ambos os lados da ampulheta necessitam interagir com apenas uma interface do GRAM, reduzindo a complexidade para a construção de novas aplicações.

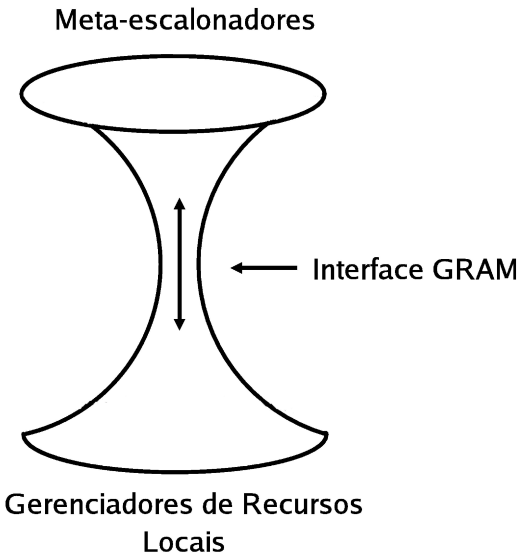

Figura 4.2: Interface implementada para a submissão de tarefas no Globus.

Todas as requisições para submissão de tarefas ao *grid* através do GRAM devem ser descritas em RSL (*Resource Specification Language*), uma linguagem que, baseada na sintaxe de filtros do LDAP (*Lightweight Directory Access Protocol*), especifica aos escalonadores detalhes de uma tarefa a ser executada. Ela inclui informações tais como o nome do arquivo executável, seus parâmetros e redirecionamento das saídas da execução. A função da RSL é padronizar todas as informações necessárias para a execução de uma aplicação (CZAJKOWSKI *et al.*, 1998; JACOB *et al.*, 2003).

A RSL possui uma sintaxe bastante simples onde cada atributo e seu valor correspondente são apresentados entre parênteses. A execução local do programa *echo*, por exemplo, que escreve na tela o que lhe for passado como parâmetro, é a seguinte:

"&(executable=/bin/echo)(arguments="Hello World!")"

A RSL possui diversos atributos tais como o para a transferência de executáveis e de arquivos necessários para a execução de uma tarefa, para a captura dos arquivos de saída remotos, para a indicação do tempo máximo de execução de um processo, assim como para a

localização do diretório base para a execução do mesmo. Em (GLOBUS, 2005) encontram-se todos os atributos e valores possíveis de serem utilizados na especificação de um comando para a submissão de tarefas utilizando-se a RSL.

A forma com que a interface de submissão de tarefas do GRAM foi implementada, possibilita a adoção de qualquer escalonador desde que os atributos da RSL sejam mapeados para os possíveis parâmetros de submissão de tarefas do escalonador que se deseja utilizar. O conjunto de ferramentas do Globus disponibiliza quatro interfaces para escalonadores: o PBS, o Condor, o LoadLeveler e o LSF.

O serviço de gerenciamento de recursos disponibilizado pelo GRAM é composto pelos seguintes programas:

- **Gatekeeper** - O gatekeeper é um processo executado na máquina responsável por autenticar e autorizar a execução de uma aplicação através do GSI e de um mapeamento do usuário identificado no GSI para um usuário local;
- **Job Manager** - O *job manager* é um processo criado pelo *gatekeeper* toda vez que uma requisição de um cliente é recebida. O *job manager* é responsável por processar as especificações enviadas em RSL pelo cliente e, na maioria das vezes, submetê-la a um escalonador local. Uma segunda funcionalidade do *job manager* é a possibilidade oferecida ao cliente de consultar o estado da aplicação submetida e cancelá-la se necessário.

A figura 4.3 ilustra o funcionamento do GRAM desde a submissão de uma aplicação pelo cliente através da interface **managed-job-globusrun**, passando pelo *gatekeer*, onde recebe a autenticação e pelo *job manager* que interpreta a RSL passada e requisita a execução da aplicação aos escalonadores locais. Os escalonadores locais, por sua vez, podem criar novos processos e distribuir as tarefas da aplicação utilizando informações obtidas pelo GRAM *Reporter* através de consultas ao serviço de informação do sistema. Uma vez criados os processos, o *job manager* monitora e controla o andamento dos mesmos até as suas finalizações. Na ilustração desse esquema, os componentes representados por linhas mais finas, isto é, o cliente GRAM e o *job manager*, são processos criados a cada requisição do usuário para a execução de uma aplicação (CZAJKOWSKI *et al.*, 1998).

Apesar dos escalonadores descritos poderem ser integrados ao Globus, eles são mais indicados para o escalonamento em *clusters*. Como o Globus pode ser utilizado para a submissão de aplicações a diferentes escalonadores, um nível mais alto de escalonamento se faz necessário para balancear a carga de trabalho entre os escalonadores locais. Tal escalonador,

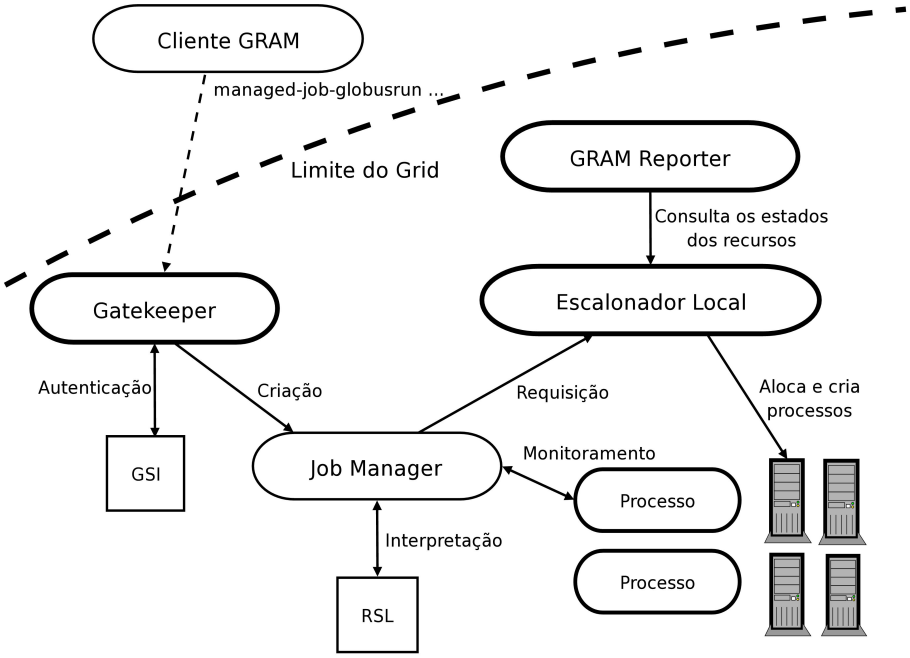

Figura 4.3: Componentes presentes no Gerenciador de Recursos do Globus.

chamado meta-escalonador, não é disponibilizado pelo Globus. Essa característica é bastante visível no momento de submissão de uma tarefa utilizando o GRAM como é mostrado a seguir:

managed-job-globusrun -factory http://lasdpc16.icmc.usp.br:8080/ogsa/services/base/gram/MasterForkManagedJobFactoryService -file test.xml

Nesse exemplo pode-se perceber a necessidade de indicação da máquina onde se deseja executar a aplicação especificada, através de comandos RSL, no arquivo test.xml. Tal medida tende a gerar resultados ruins em relação ao tempo de espera pelos resultados da aplicação, pois, muitas vezes, o usuário submete tarefas a máquinas sobrecarregadas, enquanto que podem existir máquinas que processariam a aplicação em um menor intervalo de tempo. O ideal seria a adoção de um meta-escalonador que, de acordo com os estados dos recursos, atribuísse a aplicação ao melhor recurso do momento de acordo com regras pré-estabelecidas, mais especificamente, de acordo com políticas de escalonamento.

Condor

Uma das interfaces disponibilizadas pelo Globus Toolkit utiliza o Condor, introduzido na Seção 3.7.4, como um escalonador local e que por ser orientado ao funcionamento com o Globus, recebe o nome de Condor-G (CONDOR, 2005b).

O Condor-G está dividido em duas partes: uma de gerenciamento de recursos e a outra de gerenciamento de processos. O gerenciamento de recursos é responsável por cuidar da disponibilidade das máquinas objetivando uma melhoria na execução de tarefas e uma melhor utilização dos recursos. O gerenciamento de processos, por sua vez, é o responsável por monitorar todas as etapas dos processos presentes no *grid*, desde a submissão até a conclusão da execução (JACOB *et al.*, 2003).

No Condor-G, as máquinas gerenciadoras de recursos são denominadas máquinas executoras enquanto que as máquinas gerenciadoras de processos são chamadas de máquinas de submissão.

O Condor faz uso de vantagens oferecidas pelo Globus para o acesso a múltiplos domínios. Algumas dessas vantagens consistem na autenticação, na execução remota de tarefas e na transferência de dados.

PBS

O Portable Batch System é um escalonador de aplicações em lote disponibilizado em versões comerciais e de código aberto. O PBS é composto por três componentes principais, um servidor, um executor e um escalonador (FERREIRA *et al.*, 2003a; PBS, 2005).

O servidor é o responsável pela criação, modificação, execução e monitoração dos processos do *cluster*, cabendo a apenas uma máquina esse papel.

As máquinas do *cluster* onde as tarefas são executadas recebem o nome de executoras. Por fim, o escalonamento de processos é feito pela máquina escalonadora a qual tem a capacidade de consultar os estados das máquinas executoras assim como saber quais aplicações estão à espera de processamento.

LoadLeveler

Desenvolvido pela IBM, o LoadLeveler é um escalonador de aplicações em lote capaz de balancear a carga entre máquinas pertencentes a um mesmo *cluster* (LOAD LEVELER, 2005; FERREIRA *et al.*, 2003a). Podem existir até quatro tipos de servidores em um ambiente LoadLeveler, incluindo:

- **Máquina Escalonadora** - É a responsável pela atribuição de tarefas aos nós executores;
- **Máquina Gerenciadora** - Uma única máquina do *cluster* é designada para monitorar

os estados dos outros nós e fazer o mapeamento entre as especificações de uma aplicação com as características dos recursos disponíveis;

- **Máquina Executora** - Nó onde é possível a execução de uma aplicação;
- **Máquina de Submissão** - Máquinas de submissão são capazes apenas de realizar a requisição de execução de uma tarefa e monitorar o estado de execução delas.

LSF

O *Load Sharing Facility* (LSF) é um escalonador para processamento de aplicações sequenciais e paralelas, sejam elas interativas ou em lote, com grande quantidade de processamento. Através do LSF é possível fazer a submissão remota de processos em uma rede de recursos heterogêneos de forma transparente. Uma outra vantagem é a sua compatibilidade com as especificações do OGSI, o que permite a interoperabilidade com outros sistemas (LSF, 2005).

4.4 Ambientes para o Desenvolvimento de Aplicações

Um dos problemas encontrados para a consolidação do uso de *grids* como plataformas de execução distribuída, consiste na dificuldade para o desenvolvimento de aplicações que façam uso dos serviços presentes no *grid*. Dessa forma, tem-se buscado integrar os serviços do Globus a ambientes, linguagens e *frameworks* de programação já existentes e consolidados, como, por exemplo, Java e MPI.

4.4.1 Commodity Kits

Commodity Grid Toolkits, freqüentemente referenciados por CoG kits, são formados por conjuntos de componentes que mapeiam as funcionalidades de um *grid* em ambientes e *frameworks* de *commodities* específicos. O que se busca é determinar como *commodities* e técnicas presentes em *grids* podem se integrar para aprimorar as capacidades de ambos (FERREIRA *et al.*, 2004; LASZEWSKI *et al.*, 2004, 2000; COG, 2005).

A contribuição dos *commodities* é focada na facilidade de uso e na reutilização de códigos, enquanto que a maior contribuição das tecnologias de *grid* consiste no acoplamento facilitado de recursos.

Entre os *commodities* existentes, os de maior interesse para integração com as técnicas do *grid* são Java, CORBA e Python.

4.4.2 MPICH-G2

O MPICH-G2 consiste em uma extensão do MPI versão 1.1 o qual é capaz de executar aplicações MPI utilizando os serviços do *grid* para a realização de tarefas tais como autenticação e submissão de processos (GLOBUS, 2005).

Problemas relacionados à diferença de arquitetura entre máquinas são contornados pelo MPICH-G2 através da conversão automática de dados em mensagens enviadas entre os recursos através do protocolo TCP (*Transport Communication Protocol*), automaticamente selecionado pelo MPICH-G2 caso as máquinas estejam geograficamente distantes.

4.5 Considerações Finais

Os serviços providos pelo conjunto de ferramentas Globus possibilitam a formação e a utilização de infra-estruturas altamente distribuídas, compostas por máquinas possuindo diferentes configurações, Sistemas Operacionais ou mesmo aplicações, as quais interagem de forma consistente e de maneira conhecida, por seguirem definições e especificações criadas pela *Open Grid Service Infrastructure* (OGSI) e pela *Open Grid Service Architecture* (OGSA), respectivamente. Essa característica de portabilidade atribuída aos serviços do Globus torna-se viável através da adoção de *Web services* para a invocação dos serviços prestados pelo *grid*. Por outro lado, o ganho obtido em portabilidade com a adoção de *Web services*, é perdido em eficiência devido à transmissão de dados via HTTP. Tal perda porém é, freqüentemente, aceitável para a maioria das aplicações alvo de execução em *grids* computacionais.

O código aberto do Globus, juntamente com seus serviços bem definidos, facilita o desenvolvimento de novas aplicações que fazem uso das funcionalidades desse sistema. Assim, é possível personalizar serviços presentes no *grid* tais como a autenticação de usuários e a submissão de aplicações, ou mesmo criar novos serviços os quais podem ser desenvolvidos com a ajuda de *kits* de programação como, por exemplo, os *Commodity Grid* e o MPICH-G2.

Entretanto, uma desvantagem do Globus consiste em sua interface simplificada para a submissão de aplicações a qual, por não apresentar mecanismos para a escolha das melhores máquinas executoras, não possibilita a submissão de tarefas de forma orientada à realização de um bom escalonamento. Essa medida, que tende a resultar em um mal adequado uso

dos recursos do *grid*, pode ser evitada se forem construídos serviços que incluam políticas de escalonamento para realizarem o papel do usuário na escolha das melhores máquinas para a execução de uma determinada aplicação. Dessa forma, no próximo capítulo, serão abordados os principais conceitos relativos a escalonamento de processos, focando as diferenças e necessidades existentes em plataformas compostas de *grids* computacionais. Essa abordagem permitirá entender quais são as principais características que uma política de escalonamento para aplicações *grid* deve ter para gerar melhores resultados, de acordo com interesses pré-determinados pelas partes envolvidas, sejam elas formadas por clientes ou mesmo por donos de recursos.

Capítulo 5

Escalonamento de Processos em Sistemas Distribuídos

5.1 Considerações Iniciais

O surgimento de Sistemas Multiprogramados, ainda na década de 60, mudou a forma como os processos eram computados. Usuários passaram a competir pelo uso da CPU, fazendo que os escalonadores de processos evoluíssem e conquistassem um papel importante dentro dos Sistemas Operacionais, deixando de se limitar a executar a tarefas da fila de cartões dos Sistemas em Lote (TANENBAUM, 2001; SILBERSCHATZ *et al.*, 2001).

Em um sistema com tempo compartilhado, existem diversos usuários esperando por atendimento. Sendo assim, nesses casos, é necessário o emprego de um escalonador que aplique regras direcionadas ao conjunto de processos prontos em espera e que satisfaça, dentro das condições existentes, os usuários do sistema de maneira geral. Há situações, porém, onde clientes são tratados de forma intencionalmente distinta devido à política de serviços diferenciados, que busca oferecer maior qualidade a um grupo restrito de usuários.

Dessa forma, percebe-se o porquê do escalonamento de processos ser uma área que ainda apresenta desafios para os pesquisadores. Neste capítulo, são abordados conceitos relacionados ao escalonamento de processos, tais como algoritmos e políticas, assim como os objetivos mais desejados de se obter em um bom escalonamento. Entre esses conceitos, são apresentadas características típicas do escalonamento de processos em *grids* computacionais, que, devido ao pouco tempo de existência, possui muito a ser explorado.

5.2 Objetivos de um Escalonamento de Processos

Quando os Sistemas Distribuídos surgiram e começaram a se popularizar, houve a criação de escalonadores, diferentes daqueles existentes nos Sistemas Operacionais, e que passaram a agregar a função de escolher quando e quais processos têm acesso a quais recursos em um sistema formado por um conjunto acoplado de máquinas. Por essa característica é que eles foram chamados de escalonadores globais em contraste aos escalonadores locais dos Sistemas Operacionais.

Assim, um escalonador global deve otimizar a execução de processos objetivando a satisfação de usuários que esperam pelas respostas das execuções de suas aplicações. Dessa forma, o escalonamento pode ser feito com diversas metas sendo que entre as mais utilizadas estão (JAIN, 1991; SILBERSCHATZ *et al.*, 2001; MAHESWARAN *et al.*, 1999):

- **Aumentar o *throughtput* do sistema** - *throughtput* é uma medida feita a partir do número de processos finalizados por unidade de tempo. Assim, o aumento do *throughtput*, também chamado de vazão do sistema, é facilmente obtido quando se trabalha com processos curtos por exemplo;

- **Diminuir o tempo de resposta** - consiste no intervalo entre a requisição de um usuário até a conclusão da execução da aplicação submetida, incluindo a coleta dos resultados. Essa definição de tempo de resposta inclui os tempos decorridos pelos processos da aplicação em filas de espera de recursos e tempos decorridos na execução dos mesmos;

- **Aumentar a utilização de recursos** - um escalonamento pode fazer com que recursos do sistema, tais como CPU, memória ou rede sejam utilizados ao máximo visando o uso de até 100% de suas capacidades, mesmo que para se atingir essa meta seja necessário dispensar outros critérios, tal como um alto *troughtput* por exemplo;

- **Balancear a carga do sistema** - em um sistema pode-se desejar não saturar alguns recursos enquanto outros ficam ociosos. Para isso, surge o balanceamento de carga que consiste em atribuir tarefas aos recursos de acordo com a capacidade dos mesmos.

Escalonadores globais, que daqui em diante serão referenciados apenas por escalonadores, enfrentam algumas dificuldades inexistentes em sistemas de um único processador tais como a heterogeneidade e o fraco acoplamento dos recursos, além de possíveis gargalos que se formam no escalonador caso ele necessite de conhecimento dos estados de todos os recursos

sob seu domínio. Na década de 90, surgiram resultados importantes tanto na criação de políticas de escalonamento quanto na de escalonadores de processos em Sistemas Distribuídos. No grupo de Sistemas Distribuídos e Programação Concorrente do Instituto de Ciências Matemáticas e de Computação, foram desenvolvidos trabalhos nessa área que culminaram na proposta e implementação do ambiente de escalonamento AMIGO(*Dyn**A**M**I**cal Flex**I**ble Schedulin**G** Envir**O**nment*) e na política DPWP(***D**ynamic **P**olicy **W**ithout **P**reemption*), que dá suporte a uma ampla gama de aplicações (SOUZA, 2000; ARAUJO, 1999).

5.3 Políticas de Escalonamento

A realização de um escalonamento é feita através da ativação de um conjunto de regras que ditam como e quando determinadas informações do sistema devem ser colhidas, de que maneira essas informações influenciam na distribuição de tarefas e ainda, quais serão os recursos utilizados para a execução das aplicações (SOUZA, 2000).

Durante o desenvolvivento de uma política de escalonamento, muitas vezes é preciso substituir uma boa medida de desempenho objetivando satisfazer os usuários como um todo. Pode ser desejável, por exemplo, aumentar o tempo de resposta de um processo para que mais tarefas tenham acesso aos recursos.

Uma outra característica das políticas de escalonamento é que elas devem ser focadas em um conjunto de aplicações específicas, sendo que uma política deve conhecer os detalhes das aplicações as quais irá escalonar, pois a adoção de uma política genérica pode influenciar de maneira negativa nos resultados das execuções, assim como, uma política mal aplicada que possua um conjunto de regras que não seja indicado para as características de uma certa tarefa. Em muitos casos, é mais vantajoso a utilização de políticas de escalonamento simples em vez de uma política altamente eficaz para determinados casos, mas que não atenda ao perfil do processo ao qual deve-se supervisionar.

Outro conceito freqüentemente citado é o de algoritmos de escalonamento os quais consistem em implementações de políticas de escalonamento. Sendo assim, pode-se dizer que, enquanto as políticas ditam as regras gerais de como lidar com processos e administrar recursos do sistema, os algoritmos de escalonamento estão preocupados com a implementação dessas regras, que pode ser feita de diversas formas.

5.4 Taxonomia de Casavant e Kuhl

Casavant e Kuhl, em 1988, propuseram uma taxonomia para a classificação das diversas formas de escalonamento possíveis em Sistemas Computacionais Distribuídos. A taxonomia proposta é dita hierárquica, pois ordena as características de escalonamento em grupos. A figura 5.1 ilustra a taxonomia proposta por Casavant (CASAVANT & KUHL, 1988).

Figura 5.1: Taxonomia de Casavant e Kuhl.

Com a intenção de esclarecer aspectos da taxonomia exposta pela figura 5.1, uma explicação sobre cada nível da hierarquia da classificação proposta é feita a seguir:

- **Local versus Global** - como já discutido anteriormente, o escalonamento local é aquele realizado pelo Sistema Operacional e que geralmente envolve o escalonamento de processos visando a utilização de apenas uma CPU, enquanto que o escalonamento global é feito em um sistema envolvendo a atribuição de processos a mais de um recurso, como, por exemplo, um conjunto de máquinas formadoras de um *cluster* ou *grid*.

- **Estático versus Dinâmico** - esse nível da hierarquia se dá de acordo com o momento em que o escalonamento é feito. Se ele acontecer no momento da compilação, então é dito estático, caso contrário, ocorre o chamado escalonamento dinâmico, onde só se determina para quais recursos o processo será atribuído no momento em que ele é retirado da fila de espera de processos prontos. Sendo assim, em um escalonamento dinâmico, é responsabilidade do sistema decidir onde o processo será executado durante o seu tempo de vida. Freqüentemente, principalmente quando se busca atingir um bom balancemento de carga em um sistema, escalonamentos dinâmicos possuem quatro componentes: uma política de transferência, uma política de seleção, uma política de localização e uma política de informação (SHIVARATRI *et al.*, 1992). A política de transferência é responsável por determinar se uma máquina está em condições de participar de uma migração de processos, seja para receber uma nova tarefa, seja para transferir um de seus processos em execução. Uma vez determinada uma máquina emissora de processos, deve-se escolher qual de seus processos deve ser transferido, o que é feito na política de seleção. A função da política de localização é encontrar máquinas receptoras de processos para os melhores emissores e vice-versa. Por fim, é a política de informação que determina quando e de onde devem ser colhidos dados a respeito dos estados de outras máquinas.

- **Ótimo versus Sub-ótimo** - um escalonamento ótimo só é possível se houver conhecimento *a priori* de todo o estado do sistema e da aplicação a qual será executada. Tal tarefa, porém, é, na maioria das vezes, inalcançável, pois o tempo de execução de uma aplicação, por exemplo, é muito difícil de ser estimado com precisão; além da carga dos recursos escolhidos para a execução de um tarefa também poder variar. Por essas dificuldades é que os algoritmos de escalonamento, geralmente, variam entre resultados sub-ótimos que, por sua vez, podem ser obtidos através de aproximações ou heurísticas.

- **Aproximação versus Heurística** - um algoritmo de aproximação consiste em um algoritmo que segue as mesmas regras de um algoritmo ótimo para o problema a ser solucionado. Porém, no algoritmo de aproximação, não é necessário que a solução ótima seja encontrada, bastando uma solução que atenda a uma certa aproximação definida *a priori*. Uma heurística, por sua vez, consiste de um algoritmo que leva em consideração alguns parâmetros que afetam o sistema de uma maneira indireta. Esses parâmetros, porém, são mais simples de se calcular do que os parâmetros verdadeiros para análise de desempenho por exemplo. É nesse ponto que a heurística representa uma boa alternativa para o escalonamento de processos.

- **Distribuído versus Não-distribuído** - a tarefa de escalonar processos globalmente pode ser realizada por um único processador ou por um conjunto deles. Quando a autoridade para o escalonamento é constituída de um único processador, o escalonamento é dito não-distribuído. Caso contrário, isto é, se ela provém de um conjunto de processadores, o escalonamento é dito distribuído.

- **Cooperativo versus Não-cooperativo** - a responsabilidade por escalonar processos pode ser cooperativa. Essa classificação é dada, por exemplo, quando cada processador de um sistema determina como seus recursos serão utilizados. Em um escalonamento não-cooperativo, os processadores atuam como entidades independentes que não se preocupam diretamente com nenhum objetivo que resulte na melhora de desempenho do sistema como um todo. Em vez disso, suas decisões afetam somente o desempenho local.

Apesar da classificação apresentada ser bastante ampla, ela ainda exclui algumas características muito comuns em escalonamento de processos. Essas características são bastante importantes pois introduzem novos conceitos e estão presentes em muitas políticas e, por esse motivo, algumas delas são introduzidas e analisadas na sessão 5.4.1.

5.4.1 Algoritmos Adaptativos e Probabilísticos

Um algoritmo adaptativo é classificado como sendo aquele no qual os parâmetros usados em sua implementação mudam dinamicamente de acordo com o comportamento do sistema em resposta à política de escalonamento adotada. Um algoritmo, por exemplo, que durante a execução altera o grau de importância de um parâmetro por achar que ele não está expressando corretamente o que deve ser feito, de acordo com a política de escalonamento, é dito adaptativo, ao passo que, um algoritmo que sempre considera a mesma estrutura e graus de ponderações entre os parâmetros é dito não-adaptativo.

Um outro tipo de algoritmo, chamado de probabilístico, foi muito utilizado em escalonamento de processos pela simplicidade do método implementado. A idéia consiste em, de acordo com um dado inicial, gerar aleatoriamente diversas soluções de um problema e dentre essas soluções, selecionar a melhor. Nesse método, o número de soluções geradas deve ser grande o bastante para possibilitar que, entre o conjunto apresentado, exista pelo menos uma solução que se aproxime da ótima. A vantagem desses algoritmos está na economia de tempo que seria despendido com o mapeamento de tarefas para recursos que, em muitas das vezes implica em um número grande de permutações.

5.5 Escalonamento em Grids Computacionais

Apesar de muitas políticas para Sistemas Computacionais Distribuídos serem aplicáveis a *grids* computacionais, freqüentemente, elas não oferecem bons resultados e, por isso, têm-se desenvolvido políticas de escalonamento dedicadas especialmente para esse tipo de sistema. Por trabalharem com detalhes presentes em sistemas de *grid*, essas novas políticas têm resultado em ganhos de desempenho significativos quando comparadas a políticas já existentes.

O escalonamento de processos em *grids* computacionais é uma grande área de pesquisa no campo dos Sistemas Distribuídos, pelo desafio que a própria natureza desse sistema representa (CIRNE, W., 2002; BERMAN, 1998). Exemplos de características de *grids* que representam desafios no momento da atribuição de tarefas são:

- **Grande quantidade de recursos** - A grande quantidade de recursos acarreta em problemas ao escalonador que tende a se tornar um gargalo no sistema. Esse problema torna-se maior quando a política de escalonamento adotada necessita de constantes atualizações a respeito do estado dos recursos disponíveis;
- **Grande heterogeneidade de recursos** - Devido à grande variedade de recursos que podem pertencer a um *grid*, é comum que exista muita heterogeneidade entre eles, principalmente, referentes a velocidades dos processadores, suas organizações, interconexões e Sistemas Operacionais;
- **Alto compartilhamento de recursos** - Quanto maior o número de usuários de um *grid*, maior costuma ser a variação de carga nas máquinas causadas por novos processos submetidos ao sistema. Tal característica influencia de maneira negativa nos resultados de políticas de escalonamentos que não prevêem tal alteração;
- **Movimentação e Consistência de dados** - A transferência de informações em *grids* deve ser limitada para que não haja uma degradação no desempenho causada pela baixa latência da rede de interconexão dos recursos. Sendo assim, em um *grid*, devem-se evitar aplicações que realizem muita comunicação ou que trabalhem com réplicas de dados as quais devem ser atualizadas com freqüência.

Se um *grid* acomodar muitos recursos, fica difícil para um escalonador central controlar o uso de cada dispositivo do sistema. Assim, muitas vezes, são desenvolvidos escalonadores capazes de dividir a carga de trabalho entre si cooperativamente. Dessa forma, pode-se dizer

que existem três tipos de escalonadores: centralizado, hierárquico e descentralizado (BUYYA *et al.*, 2002; BERSTIS, 2002).

Um escalonador centralizado é único para todo um sistema e, por esse motivo, representa um gargalo, pois, todas as decisões de atribuições de tarefas dependem dele. Esse tipo de escalonador é, geralmente, inviável em um *grid* devido ao grande número de máquinas que esse sistema pode acomodar. A figura 5.2 ilustra como funciona um escalonador centralizado em uma única máquina.

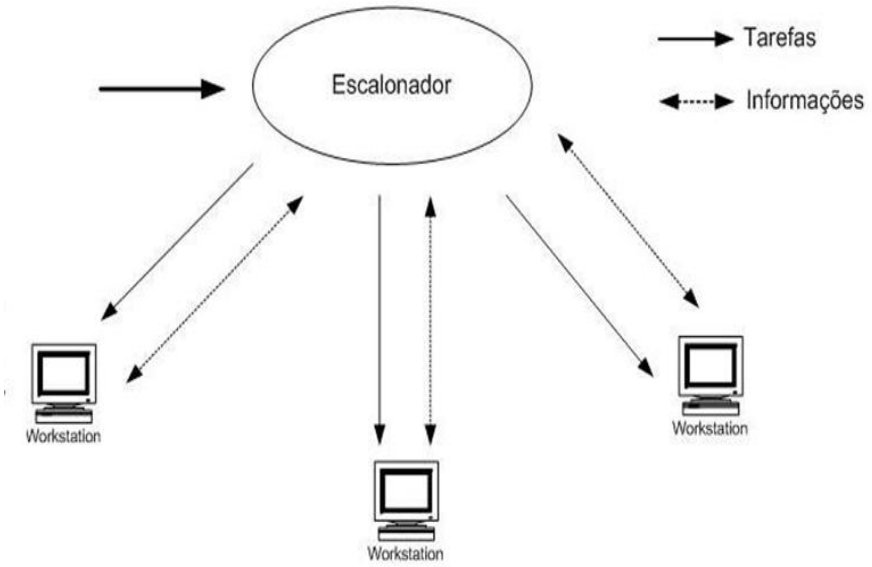

Figura 5.2: Modelo de um escalonador centralizado.

Para resolver o problema do escalonador centralizado, desenvolveu-se um modelo de escalonador que divide a carga de trabalho entre outros escalonadores do mesmo nível, formando o modelo ilustrado pela figura 5.3. Nesse modelo, existe a distribuição de autoridade entre os escalonadores, sendo que qualquer um deles pode decidir em quais máquinas um determinado processo será executado. Uma característica importante desse tipo de escalonamento é que cada escalonador tem conhecimento do estado de trabalho dos demais. Assim, quando uma requisição de execução chega a um escalonador e ele está sobrecarregado, ele verifica qual dos seus vizinhos está com menos trabalho e envia a solicitação para ele. Dessa forma, os escalonadores tornam-se balanceados com relação à quantidade de trabalho que cada um possui. Porém, esse balanceamento custa uma certa sobrecarga para que cada escalonador possua os estados dos seus vizinhos atualizados.

Um outro tipo de escalonador para *grids* é o conhecido como hierárquico, assim como

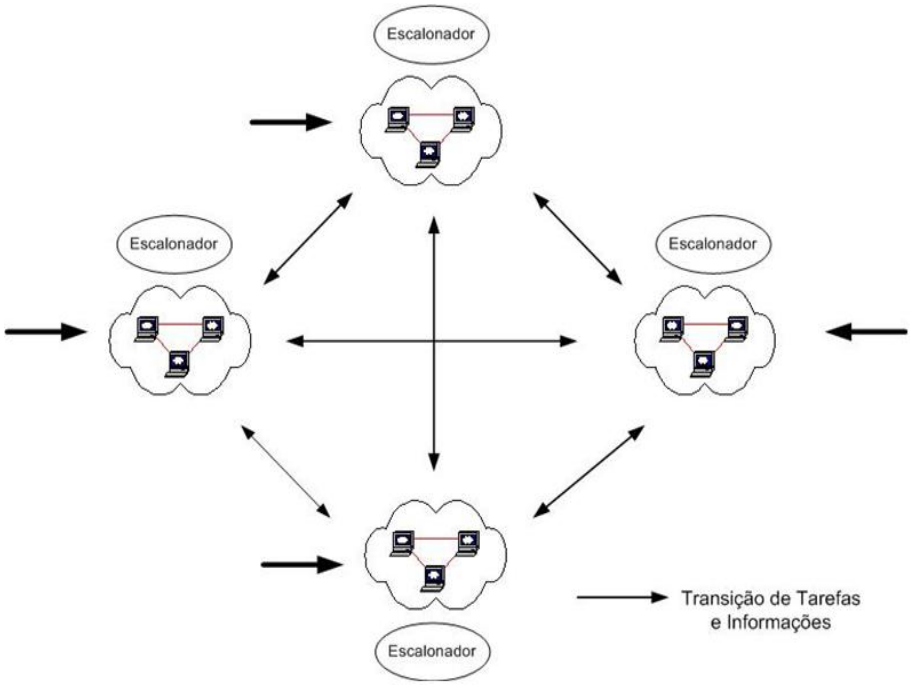

Figura 5.3: Modelo de um escalonador descentralizado.

mostra a figura 5.4. Nesse modelo, existem diversos níveis de escalonadores, onde cada escalonador, no nível mais baixo, é responsável por um subconjunto de máquinas. Nos níveis seguintes, os escalonadores, em vez de serem responsáveis por um conjunto simples de máquinas, são responsáveis por um grupo de escalonadores. Assim, a decisão de escalonamento é concentrada no escalonador de nível mais alto, porém, os demais escalonadores devem enviar periodicamente os seus estados ao escalonador superior para que ele possa realizar o escalonamento corretamente.

Quando um *grid* é formado por recursos de mais de uma organização ou domínio administrativo, um outro problema é inserido no contexto de escalonamento: como convencer o administrador do sistema local a disponibilizar seus recursos à aplicações do *grid*. Para que isso ocorra é necessário que as políticas de escalonamento postas em prática no ambiente do *grid* satisfaçam aos interesses gerais dos administradores, mesmo que para atingir esse objetivo seja necessário priorizar os processos locais deixando os processos do *grid* para serem executados em momentos de menor utilização dos recursos pelas aplicações locais. Muitas vezes, após o escalonador global do sistema escolher quais recursos utilizar, ele ainda deve enviar uma solicitação de uso aos escalonadores locais de cada domínio.

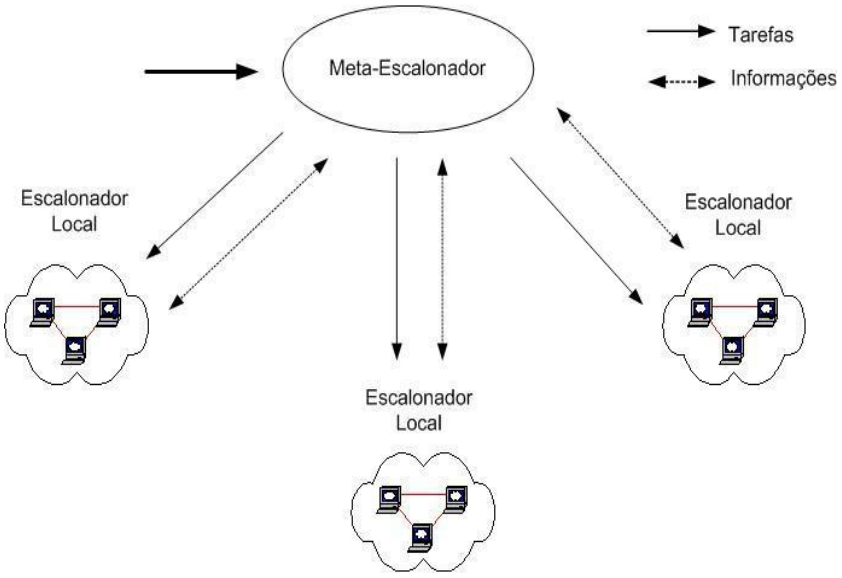

Figura 5.4: Modelo de um escalonador hierárquico.

5.6 Políticas de Escalonamento para Grids

As características de aplicações *Bag-of-tasks* implicam em uma maior simplicidade para escaloná-las, o que permite o uso de políticas baseadas somente na porcentagem de CPU livre, velocidade dessa CPU e tamanho das tarefas a serem executadas, sendo raros os casos onde considera-se a largura de banda existente entre os recursos. Políticas tradicionais, como é o caso da Workqueue e da *Round-Robin*, utilizadas localmente em Sistemas Operacionais podem, da mesma maneira, ser utilizadas para o escalonamento em *grids* computacionais. Essas políticas, apesar de serem bastante simples, representam uma grande base para o desenvolvimento de outras mais robustas e eficientes como, por exemplo, a WQR2X.

A seguir são descritas algumas das políticas mais comuns utilizadas em *grids* computacionais.

5.6.1 Round-Robin(RR)

Talvez a *Round-Robin* seja a mais simples das políticas de escalonamento, onde a atribuição das tarefas é feita de uma só vez no momento da submissão da aplicação. Nessa política, as tarefas são atribuídas aos processadores de maneira cíclica até que não haja mais tarefas não-escalonadas. Outro motivo pelo qual esse método é bastante utilizado consiste na falta de necessidade de coleta de informações do sistema, tais como tamanho das tarefas e utilização da CPU. Por outro lado, a *Round-Robin* costuma gerar grandes desbal-

anceamentos de carga entre os processadores, uma vez que, máquinas lentas recebem cargas semelhantes às máquinas mais velozes. Considerando que em um *grid* os recursos tendem a ser bastante heterogêneos e variantes, essa política freqüentemente apresenta um tempo de resposta exageradamente grande quando comparados aos de outras políticas (JAMES *et al.*, 1999).

5.6.2 Workqueue

Essa política de escalonamento atende a um *pool* de processos, onde a escolha de qual processo será submetido para execução é feita de maneira aleatória sempre que um recurso se encontrar disponível. Assim como na *Round-Robin*, essa política não necessita de informações do sistema ou da aplicação, porém, ela apresenta um tempo de resposta menor pois atribui dinamicamente as tarefas aos processadores que ao passar do tempo se tornam livres. Assim, processadores mais velozes tendem a se tornar disponíveis mais rapidamente e, conseqüentemente, a receber mais tarefas. Contudo, o problema de desbalanceamento ainda persiste caso uma grande tarefa seja atribuída a um processador lento perto do final da execução da aplicação (SILVA, 2003).

5.6.3 Max-Min

Baseada na idéia de atribuir as maiores tarefas para as melhores máquinas, essa política consegue baixos tempos de resposta quanto maior for o número de pequenas tarefas em relação às grandes. Isso porque quando as tarefas maiores tiverem desbalanceado o sistema, as tarefas menores são executadas nas máquinas mais velozes enquanto que, as máquinas mais lentas finalizam o processamento das grandes tarefas que receberam (MAHESWARAN *et al.*, 1999; CASANOVA *et al.*, 2000).

Ao contrário da *Round-Robin* e da Workqueue, na Max-Min são necessárias três informações sobre a aplicação a ser executada e sobre os recursos disponíveis para execução, chamados de *hosts*. São elas:

- ***Task_Size*** - expressa o tamanho total de instruções da tarefa em MI (Milhões de Instruções);
- ***Host_Load*** - representa a porção do recurso que está sendo utilizada e, por esse motivo, não está disponível para uso do *grid*;

- ***Host_Speed*** - representa um valor relativo à velocidade da máquina em MI por segundo.

Para a alocação de recursos, os processos são ordenados pelos seus tamanhos de forma descendente. Assim, os processos maiores serão os primeiros a serem atendidos, de forma que, busca-se atribuí-los aos processadores que oferecem os menores tempos para completar a execução desses processos. Esse tempo é chamado de *Completion Time*(CT) e é calculado através da fórmula:

$$\text{CT} = \text{TBA} + \text{Task\_Cost}$$

onde a variável Task_Cost é representada por:

$$\text{Task\_Cost} = (\text{Task\_Size}/\text{Host\_Speed})/(1 - \text{Host\_Load})$$

A varíavel TBA(*Time to Become Available*) representa o tempo necessário para que um determinado recurso esteja disponível para o sistema. Inicialmente, todos os *hosts* têm seu TBA zerado. Quando um *host* recebe uma tarefa, ele tem seu TBA incrementado de Task_Cost.

Por ser uma política estática, já que todo o mapeamento entre tarefas e recursos é feito no momento da submissão da aplicação, a Max-Min escalona mal quando há muita variação na carga do sistema, o que, infelizmente, é um acontecimento freqüente em *grids* computacionais.

5.6.4 Min-min

Ao contrário da política Max-Min, a Min-Min atribui as menores tarefas às melhores máquinas, o que tende a resultar em um maior desbalanceamente de carga entre os recursos quando comparada à Max-Min. Pode-se dizer, porém, que a política Min-Min visa um alto *throughtput*, pois os melhores recursos são liberados mais rapidamente, podendo assim executar novas tarefas (MAHESWARAN *et al.*, 1999; CASANOVA *et al.*, 2000).

5.6.5 Dynamic FPLTF

Essa política consiste em uma extensão da FPLTF(*Fastest Processor to Largest Task First*), onde as maiores aplicações, isto é, aquelas que requerem maior tempo de CPU, são atribuídas aos processadores mais velozes, porém, de uma forma dinâmica.

Na DFPLTF, quando uma tarefa é finalizada, os recursos alocados por ela tornam-se disponíveis para as outras tarefas que ainda não iniciaram execução e que serão escalonadas novamente, contornando assim, qualquer variação de carga existente nos recursos (SILVA, 2003).

5.6.6 Sufferage

Essa política, assim como a Max-Min, Min-Min e *Dynamic FPLTF* exige as informações de *Task_Size*, *Host_Load* e *Host_Speed*. A Sufferage, porém, consiste na idéia de atribuir a melhor máquina à aplicação que mais seria prejudicada se essa atribuição não fosse feita. A diferença entre o melhor e o segundo melhor tempo para completar a tarefa, chamada de *sufferage*, é que mede o quanto a aplicação irá perder, ou sofrer de acordo com a nomenclatura da política. Assim, a tendência é que uma tarefa seja atribuída à máquina que oferece o menor tempo para completá-la (CASANOVA *et al.*, 2000; MAHESWARAN *et al.*, 1999; SILVA, 2003).

Toda vez que um processo termina de ser executado, as tarefas em espera por recursos têm seus valores de *sufferage* recalculados e, de acordo com eles, são reescalonadas pelo sistema. Esses passos são seguidos até que não haja mais nenhuma aplicação esperando pela sua execução.

5.6.7 Workqueue with Replication

Essa política é uma extensão do *Workqueue* tradicional onde uma coleção de processos é criada e atendida aleatoriamente de acordo com a disponibilidade de recursos existentes. Assim como a Workqueue, ele se difere das duas políticas anteriormente descritas por não necessitar de informações do sistema ou mesmo da aplicação e essa é a sua grande vantagem em relação às demais. Na *Workqueue with Replication* (WQR), quando a fila de processos estiver vazia e existirem processadores ociosos, tarefas de uma mesma aplicação em execução são escolhidas para ser replicadas nesses processadores (CIRNE, W., 2002; SILVA, 2003).

Quando uma réplica termina de ser executada, todas as outras são abortadas para que não haja desperdício de processamento. Essa técnica é baseada na idéia de que o tempo necessário para completar uma tarefa que foi replicada possui grande possibilidade de ser reduzido por alguma das réplicas.

5.7 Considerações Finais

O escalonamento de processos em Sistemas Distribuídos, apesar de ser um tema que vem sendo há muito tempo estudado, ainda representa uma tarefa desafiadora. Em um *grid* computacional, em especial, esse desafio é maior devido à natureza que esse sistema pode representar, tal como uma grande extensão geográfica, grande quantidade de recursos com utilização variável, além de, em alguns casos, serem formados por diversas instituições cada qual possuindo sua própria política de uso.

A grande variação da carga dos processadores formadores do *grid* faz com que políticas de escalonamento dinâmicas sejam as mais adequadas para uso nesse tipo de sistema (JAMES *et al.*, 1999). Políticas tais como a *Round-Robin*, a Workqueue e mesmo a Max-Min, não costumam apresentar bons resultados por falta de acesso a informações a respeito das aplicações a serem executadas ou mesmo dos atuais estados dos recursos existentes no *grid*.

As políticas DFPLTF e Sufferage, por sua vez, utilizam informações das aplicações e dos recursos (freqüentemente atualizadas), sendo que, desse modo, tendem a realizar um melhor escalonamento de processos. O uso dessas informações, porém, pode acarretar dificuldades pois, muitas vezes, a coleta de informações imprecisas prejudica bastante o desempenho de políticas de escalonamento, principalmente quando a atribuição de tarefas é feita de forma estática tal como na Max-Min (CASANOVA *et al.*, 2000; BERMAN, 1998). Para contornar esse problema, outras políticas como, por exemplo, a *Workqueue with Replication*, conseguem bons resultados na execução de suas aplicações mesmo sem utilizar dados sobre as mesmas. Essa facilidade pode custar alguma perda de ciclos de CPU e do desempenho das aplicações, mas representa uma vantagem quando se trata de comodidade e transparência. Assim, a escolha de qual política utilizar depende bastante dos interesses dos usuários que devem estar de acordo com as regras dos proprietários dos recursos, devendo ser estudado criteriosamente para que nenhuma das partes seja desfavorecida.

O próximo capítulo apresentará a proposta de uma nova política de escalonamento, chamada de *Dynamic* Max-Min2x a qual objetiva a diminuição do tempo de resposta de uma aplicação usando das melhores características presentes em algumas políticas de escalonamento de processos descritas neste capítulo.

Capítulo 6

A Proposta de Novas Formas de Submissão de Tarefas em um Grid

6.1 Considerações Iniciais

A adoção de políticas orientadas ao escalonamento de tarefas em *grids* computacionais tende a melhorar, significativamente, o desempenho de aplicações executadas nesse tipo de plataforma. Porém, o sistema para construção de *grids* mais utilizado, o Globus, não disponibiliza uma forma para o escalonamento de processos utilizando-se de informações globais do sistema, isto é, não é possível, por meio dos serviços básicos presentes no Globus, a realização de um meta-escalonamento. Por outro lado, o Globus permite que seus serviços sejam utilizados na criação de novas aplicações, o que facilita o desenvolvimento de projetos que atendam às necessidades de cada usuário. Dessa forma, este capítulo apresenta a proposta da construção de um *framework* dinâmico para a submissão inteligente de tarefas ao *grid*. Esse *framework* é construído utilizando-se serviços presentes no Globus e implementados com a ajuda do Java CoG Kit.

Ainda neste capítulo, é apresentada a concepção de uma nova política de escalonamento para aplicações do tipo *Bag-of-Tasks*, a *Dynamic* Max-Min2x. A importância dessa política consiste na redução do tempo de resposta necessário para um usuário receber os resultados da execução de sua aplicação. Para finalizar, neste mesmo capítulo, são descritas as formas para a avaliação da *Dynamic* Max-Min2x e os resultados obtidos em testes com outras políticas de escalonamento freqüentemente utilizadas.

6.2 Um Framework para Submissão de Tarefas no Grid

Através da interface para programação disponibilizada pelo Globus (COG, 2005; LASZEWSKI *et al.*, 2004, 2000), foi possível a implementação de uma aplicação para submissão de tarefas em um *grid* computacional. Para isso, foram utilizados os serviços de:

- Segurança - a autenticação e autorização de usuários, assim como a transferência de dados entre computadores, é realizada com o uso de chaves públicas e certificados presentes na infra-estrutura provida pelo GSI;
- Gerenciamento de dados - a transferência de executáveis e arquivos de entrada/saída é feita utilizando-se os serviços do GASS, onde dados são transferidos entre máquinas através do protocolo HTTP;
- Gerenciamento de recursos - a submissão de tarefas ao *grid*, assim como o monitoramento da execução dos mesmos, é implementado utilizando-se as funcionalidades presentes no GRAM.

O serviço de descoberta de recursos foi implementado utilizando-se informações obtidas a partir do NWS (*Network Weather Service*) (NWS, 2005; WOLSKI *et al.*, 1999), um sistema que periodicamente realiza o monitoramento de recursos computacionais (CPU, memória, rede) e dinamicamente prevê o desempenho dos mesmos em um curto intervalo de tempo. Essas informações são disponibilizadas através de *Web services*, para que a interoperabilidade do sistema não seja comprometida.

A figura 6.1 ilustra o funcionamento da aplicação por etapas. Na primeira fase, o cliente faz uma consulta a um servidor responsável por armazenar todos os recursos disponíveis no sistema. A resposta, consistindo em uma lista com o endereço de todos os recursos, é utilizada pelo cliente para consultar tais recursos e obter informações sobre o estado de cada um deles, como, por exemplo, carga da CPU e porcentagem de memória não-paginada.

As informações colhidas são muito importantes, pois servem como parâmetros para as políticas de escalonamento em uso que, por sua vez, através da aplicação de regras, estabelecem quais os recursos mais aptos para processar as tarefas do cliente. Por fim, como exemplificado pela figura 6.2, a transferência dos dados necessários para execução das tarefas é feita e, em seguida, o processamento das mesmas. Caso a aplicação a ser submetida não realize comunicação entre suas tarefas, então para cada tarefa são criadas

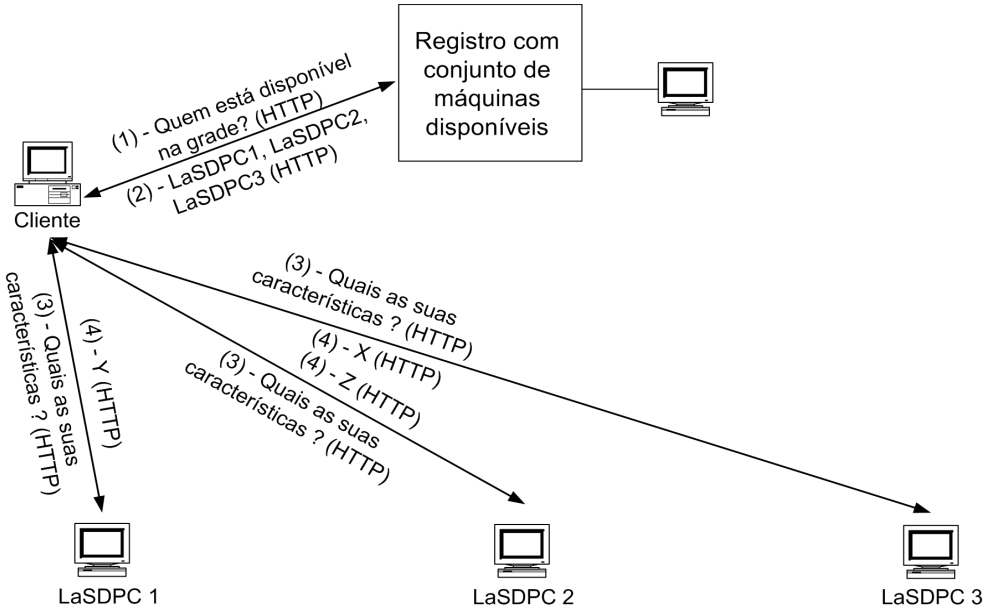

Figura 6.1: Etapas para aquisição de informações do *grid*.

threads responsáveis por monitorar a execução de seu processo. Caso contrário, é proposto que ela seja submetida para um conjunto de máquinas onde exista o PVM/MPI executando o ambiente de escalonamento AMIGO (*Dyn**AM**ical Flex**I**ble Schedulin**G** Envir**O**nment*).

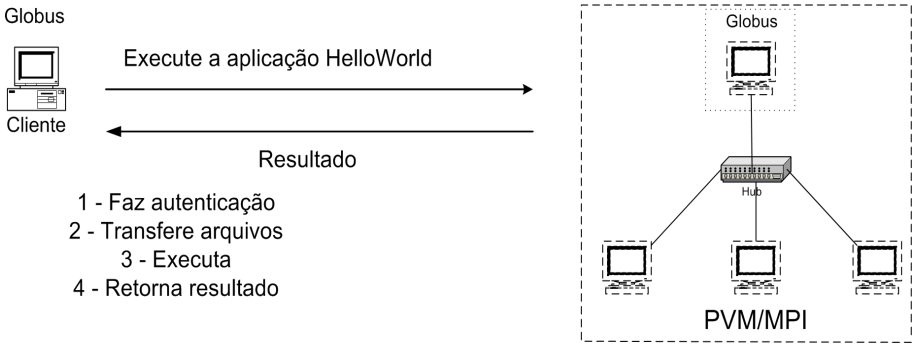

Figura 6.2: Passos da submissão de uma tarefa.

O AMIGO é uma ferramenta responsável pela gerência do escalonamento de processos em plataformas computacionais distribuídas. Isso porque ele possibilita a inclusão de políticas de escalonamento as quais podem ser focadas em diferentes objetivos, de acordo com o determinado pelo usuário (SOUZA, 2000).

O termo dinâmico refere-se à característica do AMIGO de alterar a política de escalonamento em uso em tempo de execução. A intenção é que mudanças no escalonamento sejam realizadas quando os resultados obtidos em monitoramentos sejam insatisfatórios. Essa possibilidade de alteração na forma de atribuição de tarefas aos processadores atribui flexi-

bilidade ao ambiente e é realizada de forma transparente ao usuário.

O AMIGO está dividido em duas camadas independentes: a camada superior, responsável pela configuração do sistema e monitoramento dos processos; e a camada inferior, responsável pelo escalonamento dos processos e constituída de três módulos, sendo eles, o AMIGOD (AMIGO *Daemon*), as políticas de escalonamento e o ambiente de passagem de mensagem. Atualmente, existem duas versões do AMIGO para ambientes de passagem de mensagem, sendo uma para PVM (*Parallel Virtual Machine*) e a outra para MPI (*Message Passing Interface*). Essas versões têm o seu código alterado para chamarem às políticas de escalonamento inseridas no AMIGO ao invés da política padrão adotada por elas: a *Round-Robin*.

O mapeamento da linguagem RSL do Globus para os comandos do AMIGO consistiu apenas na inclusão do comando *pvm* na RSL e pequenas alterações dos arquivos fontes. Isso porque a forma que o AMIGO foi desenvolvido, prezando pela transparência, não necessita de nenhum novo comando além dos conhecidos pelo MPI e pelo PVM, sendo que o MPI já possui o mapeamento de seus comandos nas distribuições do Globus. A figura 6.3 ilustra os passos para a execução de uma tarefa que utilize o MPI com o AMIGO *daemon* rodando nas máquinas executoras. No primeiro passo, os atributos RSL presentes no arquivo XML de entrada são transformados pelo *job manager* em uma expressão a qual através de um comando *fork* é executada como um novo processo na máquina. Quando o processo MPI em execução realiza o escalonamento de suas tarefas, o AMIGO *daemon* o intercepta evitando que as tarefas sejam distribuídas de acordo com a política padrão do MPI, aplicando em seu lugar a política que, no momento, se encontrar ativa no ambiente de escalonamento.

A proposta do *framework* para submissão de tarefas em questão é especialmente importante em um *grid* onde há muitos recursos a serem utilizados. Tais recursos, se explorados de maneira correta, podem oferecer melhores desempenhos aos usuários do sistema. Sendo assim, a seção 6.3 apresentará a proposta de uma nova política que, utilizando-se de replicações, objetiva a redução do tempo de resposta de aplicações cujas tarefas não necessitam de comunicação entre si. Essa política, se aplicada em um *grid* no momento da submissão de tarefas pode apresentar ganhos significativos se comparada a outras políticas freqüentemente em uso em plataformas de execução tais como a de um *grid*.

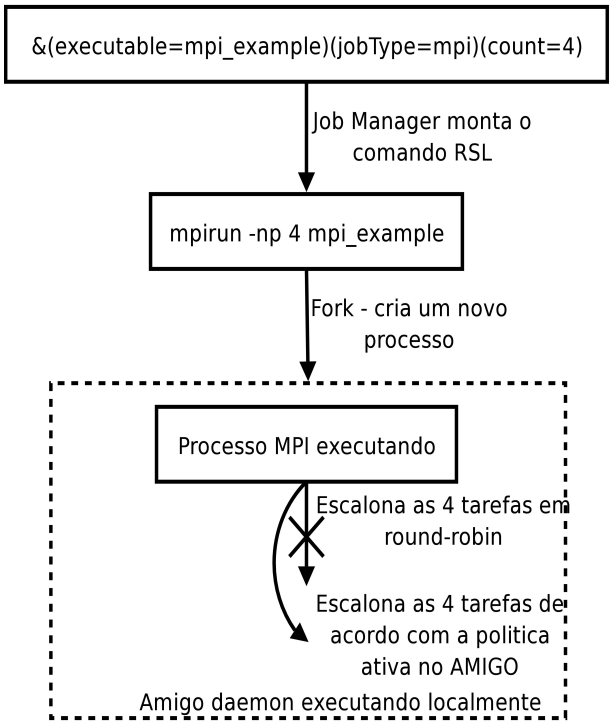

Figura 6.3: Passos da submissão de uma tarefa em máquinas com o AMIGO *daemon* sendo executado.

6.3 Dynamic Max-Min2x

A variação de carga nos recursos utilizados para a execução de uma aplicação podem afetar de maneira bastante negativa o desempenho obtido caso a política de escalonamento seja realizada de maneira estática. Ignorar essa variação e atribuir tarefas dinamicamente ao *grid* também mostra-se igualmente ineficaz. Por outro lado, a atualização do estado dos recursos a cada atribuição de tarefa, causa uma sobrecarga na comunicação do sistema.

Dessa forma, é proposta a *Dynamic* Max-Min2x, uma política de escalonamento para aplicações do tipo *Bag-of-Tasks* baseada na atribuição dinâmica de tarefas ao *grid*. Ela recebe a extensão 2x pois toda tarefa escalonada pela *Dynamic* Max-Min2x tem a possibilidade de ser executada em duas máquinas, isto é, a *Dynamic* Max-Min2x é uma política com replicação de tarefas.

Assim como na política Max-Min, a DMax-Min2x, como será chamada a política *Dynamic* Max-Min2x de agora em diante, reduz o desbalanceamento de carga entre os processadores através da aplicação de maiores prioridades às maiores tarefas, mas difere dela ao atribuir dinamicamente as tarefas. Assim, as partições de uma aplicação são escalonadas conforme os processadores se tornam disponíveis. Esse mecanismo, por distribuir um maior

número de tarefas aos recursos com maior capacidade, os quais finalizam os processamentos mais rapidamente, tende a apresentar um melhor tempo de resposta para as aplicações.

Um problema presente na política *Dynamic* Max-Min consiste na obtenção de informações a respeito das tarefas a serem submetidas ao *grid* como, por exemplo, o tamanho das tarefas, sendo que uma estimativa mal feita pode degradar o seu desempenho. Para contornar essa situação, pode-se aplicar mecanismos que utilizam-se de históricos de execução para a classificação e extração de informações a respeito das aplicações que serão submetidas ao sistema. Estudos têm demonstrado sucesso na obtenção de informações com a utilização desses mecanismos (SENGER *et al.*, 2004).

Outra característica da DMax-Min2x é que ela utiliza replicações de tarefas como um meio para a redução do tempo de resposta, assim como a WQR2x. Como a DMax-Min consiste de uma política dinâmica, ela tende a igualar as cargas de trabalho entre os processadores, o que faz que eles terminem os seus respectivos processamentos em intervalos de tempo similares o que gera, como conseqüência, um pequeno desperdício de ciclos de CPU quando comparado ao número de ciclos utilizados em replicações de políticas estáticas.

Pode-se dizer, portanto, que a DMax-Min2x é baseada nas principais características das políticas Max-Min, DFPLTF e WQR2x, que consistem em: atribuição das maiores tarefas aos melhores processadores, dinamicidade do escalonamento e replicação de tarefas. A tabela 6.1 apresenta uma síntese das características presentes nessas políticas.

Políticas	Informações de Tarefas	Informações de Recursos	Dinamicidade	Resposta à mudança de carga	Replicação
Workqueue	NÃO	NÃO	SIM	RUIM	NÃO
WQR2x	NÃO	NÃO	SIM	MÉDIA	SIM
DFPLTF	ALTA	ALTA	SIM	BOA	NÃO
Max-Min	ALTA	ALTA	NÃO	RUIM	NÃO
DMax-Min2x	BAIXA	BAIXA	SIM	BOA	SIM

Tabela 6.1: Principais características presentes nas Políticas de Escalonamento para *grids*.

A política DMax-Min2x consiste em uma heurística de escalonamento global e dinâmica, a qual é descrita através do algoritmo 1.

A DMax-Min2x, através de simulações, mostrou atingir um desempenho superior à DFPLTF mesmo sem a necessidade de atualização dos estados dos recursos do *grid*. A DMax-Min2x também apresentou resultados significativos quando comparados à outras políticas tais como a WQR2x e a Max-Min.

Algorithm 1: Pseudo-código da heurística utilizada pela política DMax-Min2x

Ordene, em uma fila T, as tarefas em ordem decrescente de tamanho (MI).
Marque todas as tarefas de T como não-replicadas.
Ordene, em uma fila R, os processadores em ordem decrescente de capacidade (MI/s).
while (houver processador em R) **do**
  Atribua a tarefa t_0 a r_0, onde t_0 e r_0 são os primeiros elementos de T e R, respectivamente.
  Remova t_0 de T.
  Remova r_0 de R.
  Insira t_0 na fila de tarefas em execução E.
end while
while (houver tarefas em T) **do**
  Espere até que um processador P torne-se disponível.
  Retire a tarefa que P estava processando da fila de tarefas em execução E.
  Atribua t_0 a P.
  Insira t_0 na fila de tarefas em execução E.
end while
while (houver tarefas não replicadas na fila de execução E) **do**
  Espere até que um processador P torne-se disponível.
  Retire a tarefa que P estava processando da fila de tarefas em execução E.
  Se a tarefa retirada de E é replicada, então cancelar a execução da outra réplica.
  Atribuir a P a primeira tarefa de E que não possua cópias e marcá-la como replicada.
end while

6.4 Avaliação da DMax-Min2x

A partir da proposta de uma nova política de escalonamento, é preciso que experimentos sejam realizados com o objetivo de avaliá-la, isto é, de mostrar que a política realmente funciona em sistemas reais. Existem dois métodos para a avaliação de políticas de escalonamento: a execução em plataformas reais e a utilização de simulações que representem de forma semelhante um sistema em *grid* (LAW & KELTON, 1982; FEITELSON & RUDOLPH, 1998).

O primeiro método consiste na implantação da política a ser analisada em um infra-estrutura de *grid* real. Essa abordagem introduz um grande obstáculo que consiste na dificuldade de acesso a uma infra-estrutura de *grid* real para a realização de testes.

Uma segunda alternativa é a utilização de simulações que fazem uso de modelos para reproduzir o comportamento de um sistema em *grid* real. Esse método, muito usado para a avaliação de políticas de escalonamento em máquinas paralelas, ainda representa uma área de pesquisa em sistemas de *grid*.

Algumas vantagens do uso de simulações na avaliação de políticas de escalonamento são:

- **Baixo custo** - Simulações consistem de programas que, freqüentemente, podem ser executados em recursos de fácil acesso;
- **Flexibilidade de cenário** - Em simulações, diferentes cenários para a execução de aplicações podem ser construídos através da atribuição de diferentes valores aos parâmetros do programa. Tal flexibilidade é inexistente em *grids* computacionais reais, onde os recursos são permanentes e a carga de trabalho das máquinas é pouco controlável;
- **Repetição de experimentos** - Como a determinação de um cenário de simulação se dá através dos parâmetros de entrada do programa, é possível a reprodução de uma determinada configurações de *grid* através da inserção de um mesmo conjunto de valores. Essa característica é especialmente importante no desenvolvimento de políticas de escalonamento, pois possibilita a análise, sob uma mesma condição, de diferentes formas de escalonamento.

Por outro lado, simulações representam sempre uma análise aproximada, isto é, uma estimativa do comportamento real de um sistema. Dessa forma, diversas execuções de um mesmo cenário devem ser repetidas para se atinja uma confiança maior nos resultados da simulação. Essa necessidade, muitas vezes, faz com que a simulação consuma um tempo muito grande para atingir um determinado grau de confiabilidade.

Para a avaliação de políticas através de simulação ser aceita, é preciso a reprodução de diversos cenários para estudo dos comportamentos resultantes dessas variações. Em *grids* computacionais, cenários potenciais para o estudo de políticas de escalonamento devem apresentar cargas diferentes no sistema (computadores ociosos e/ou saturados), diferentes variações nessa carga, diferentes graus de heterogeneidade entre os recursos, diferentes tamanhos, tipos e heterogeneidade de tarefas, entre muitos outros parâmetros a depender da finalidade da política proposta.

A variação na carga dos recursos, em especial, é uma característica que tem sido freqüentemente ignorada em simulações de *grids* computacionais. Isso devido à dificuldade de se expressar tal comportamento por falta de um estudo mais abrangente em tais plataformas. Essa medida, porém, gera dúvidas quanto à validade e, por conseqüência, limita a confiabilidade da política sendo analisada. Uma alternativa é a utilização de arquivos, chamados

traces, contendo medidas colhidas de *grids* computacionais reais. Essa abordagem apesar de mais próxima do comportamento real, ainda é questionável pois pode apresentar características muito particulares de determinados sistemas. Para contornar esse problema, é importante a análise de um alto número de *traces* com intervalos de tempo longos o bastante para expressar o maior número de cenários possíveis.

6.5 Simulações para Avaliação da DMax-Min2x

A avaliação da política DMax-Min2x foi feita através de simulações, sendo que dois modelos foram utilizados para que a política fosse avaliada sob diferentes aspectos em relação à carga de trabalho. Uma das simulações tem a carga dos processadores baseada em *traces*, enquanto que a outra tem a carga dos processadores baseada em um modelo de carga adaptado para uso em *grids* (SILVA, 2003; MELLO & SENGER, 2004).

É importante lembrar que nessas simulações foram consideradas apenas aplicações do tipo *Bag-of-Tasks*, isto é, que fazem uso de pouca entrada/saída de dados, uma vez que os tempos para a transferência de dados pela rede são ignorados. Além disso, é assumido que as aplicações não possuem nenhum prazo máximo de execução.

6.5.1 Simulação Baseada em Traces

No primeiro modelo de simulação, são utilizados *traces* coletados de 116 máquinas de uma universidade[1]. Esses *traces* consistem de arquivos formados por duas colunas: a primeira sendo um *timestamp* e a segunda a porcentagem de CPU disponível naquele *timestamp* assim como ilustra a figura 6.4.

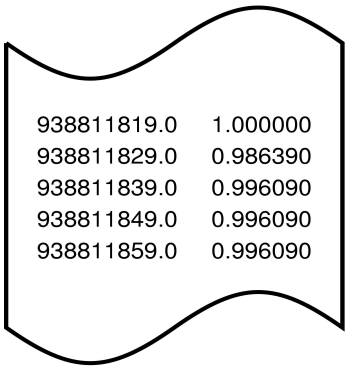

Figura 6.4: Formato dos arquivos de *traces* utilizados na simulação.

[1]Traces coletados na Universidade da Califórnia.

Nesse modelo, as simulações para avaliação da DMax-Min2x foram desenvolvidas com o auxílio do SimJava, um pacote de simulação orientado a eventos que conta com as vantagens providas pela linguagem Java, tais como a independência de plataforma e a orientação a objetos (SIMJAVA, 2005).

Os experimentos realizados neste trabalho têm como objetivo a análise do comportamento da DMax-Min2x sob diversas condições. Para isso são simulados cenários com diferentes tamanhos e heterogeneidade de tarefas e diferentes heterogeneidade de recursos.

Simulação das Aplicações

A modelagem das aplicações foi determinada para que a alta relação de máquinas por tarefas e a saturação dessas máquinas (muitas tarefas para poucas máquinas) pudessem ser analisadas. Dessa forma, como ilustrado pela tabela 6.2, cada aplicação possui tamanho 3600000 Milhões de Instruções (MI) que é dividida grupos de tarefas de tamanho médio 1000, 5000, 25000 e 125000 MI. Quanto menor o tamanho médio das tarefas, maior o número delas e, conseqüentemente, maior a quantidade de tarefas que cada máquina tende a processar.

Tamanho Médio das Tarefas (MI)	Quantidade de Tarefas	Máquinas por Tarefa	Tarefas por Máquina
1000	3600	0,028	36
5000	720	0,14	7,2
25000	144	0,7	1,44
125000	29	3,45	0,29

Tabela 6.2: Granulosidade das Aplicações.

Dentro de um mesmo grupo é possível existir heterogeneidade entre as tarefas. Tal heterogeneidade pode ser de 0, 25, 50, 75 ou 100%, porém, cada grupo de tarefas tem uma média de tamanho que permanece constante. Isso porque o tamanho das tarefas é determinado por uma distribuição uniforme **U(lim_inferior, lim_superior)**. Por exemplo, tarefas de tamanho 1000 MI e heterogeneidade de 25%, tem a distribuição ditada por **U(875,1125)**, onde 875 e 1125 consistem nos limites inferiores e superiores dos tamanhos das tarefas do grupo, respectivamente.

Simulação dos Recursos

A heterogeneidade de recursos é bastante presente em *grids* formados por computadores pessoais. Para representar essa heterogeneidade, foi utilizado o conceito da Lei de

Moore que diz que a cada 18 meses, os computadores têm a sua capacidade de processamento dobrada. Dessa maneira, considerou-se a diferença de 6 anos na idade dos recursos do *grid* simulado, o que gera computadores com até 16 vezes mais potência que os computadores mais lentos da plataforma[2].

Assim como na simulação das aplicações, na simulação dos recursos a distribuição uniforme também é utilizada, porém, dessa vez, a média gira sempre em torno do mesmo valor de capacidade dos recursos: 10 MI, que varia de acordo com as heterogeneidades 1, 2, 4, 8 e 16. Sendo assim, um *grid* com recursos possuindo heterogeneidade 8 são determinados pela distribuição **U(6,14)**, onde 6 é a menor e 14 a maior capacidade atingida por um recurso desse *grid*.

Como todos os cenários da simulação possuem recursos cujas capacidades variam em torno da média 10 MI, então em todos os *grids* simulados existem cerca de 100 máquinas disponíveis para execução de aplicações.

Simulação das Cargas de Trabalho dos Recursos

A carga dos recursos é dirigida de acordo com *traces* cujas informações a respeito da utilização de CPU são coletadas, em média, a cada 10 segundos com o uso do NWS (*Network Weather Service*).

Na simulação, o comportamento de cada computador é guiado por um dos arquivos de *trace* escolhido aleatoriamente entre os diversos arquivos disponíveis. Dessa forma, supondo que o computador A tenha seu estado baseado no *trace* da figura 6.4, ele começa a simulação com 100% de sua capacidade de processamento de CPU livre. Passados 10 segundos, essa capacidade é alterada para 98,64%.

Análise dos Resultados

Os experimentos deste trabalho foram realizados com o objetivo de comparar os resultados obtidos pela DMax-Min2x com as políticas de escalonamento WorkQueue, Max-Min, DFPLTF e a WQR2x.

Foram simulados 100 cenários distintos (4 tamanhos de tarefas cada uma com 5 níveis de heterogeneidade e 5 níveis de heterogeneidade de recursos), sendo que cada cenário foi simulado até que se atingisse um intervalo de confiança de 95%.

[2]Considerando-se inicialmente, que processadores lançados no mercado alcançam desempenho x, os lançamentos de processadores em um intervalo de tempo de 18, 36, 48 e 72 meses tendem a apresentar um desempenho de 2x, 4x, 8x e 16x, respectivamente.

Através de simulações, é possível perceber que a política DMax-Min2x obtém uma média de resultados superior às das outras políticas. Análises mais específicas, porém, revelam detalhes importantes a respeito desses resultados e mostram que as vantagens da DMax-Min2x podem ser maiores em determinadas condições. Dessa forma, as seções a seguir têm a finalidade de apresentar análises dos resultados obtidos nas simulações de forma mais específica, através da divisão de cenários.

Comportamento das Políticas com Diferentes Tamanhos de Tarefas

A granulosidade das tarefas de uma aplicação mostra-se como um parâmetro determinante na escolha de uma política de escalonamento adequada assim como ilustra a figura 6.5, onde cada ponto das linhas representa a média aritmética dos tempos de execução nos cinco níveis de heterogeneidade de máquinas e nos cinco níveis de heterogeneidade de tarefas.

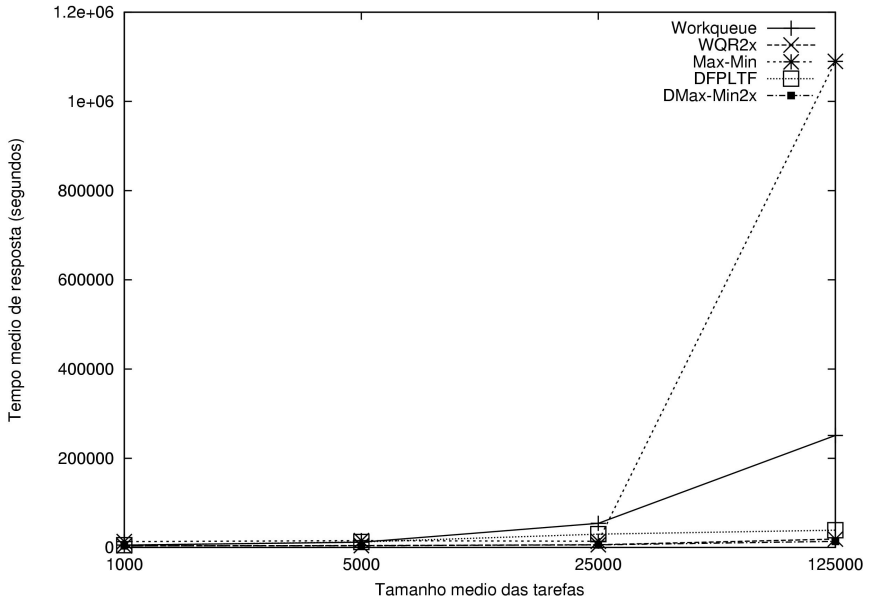

Figura 6.5: Desempenho das políticas de acordo com os diferentes tamanhos de tarefas.

A figura 6.6 ilustra os resultados sem considerar o desempenho da política MaxMin para tarefas com média de tamanho igual a 125000 MI pois ela apresenta resultados bastante discrepantes o que atrapalha a visualização dos gráficos.

A análise dos gráficos mostra que quanto maior a relação do número de tarefas pelo número de máquinas (menor granulosidade das tarefas), mais fácil torna-se o escalonamento para as políticas Workqueue e a WQR2x, que não utilizam informações do sistema, e mais semelhantes são os resultados dessas políticas em relação à política DMax-Min2x, que necessita de dados sobre as tarefas e os recursos disponíveis. A política Max-Min, por sua

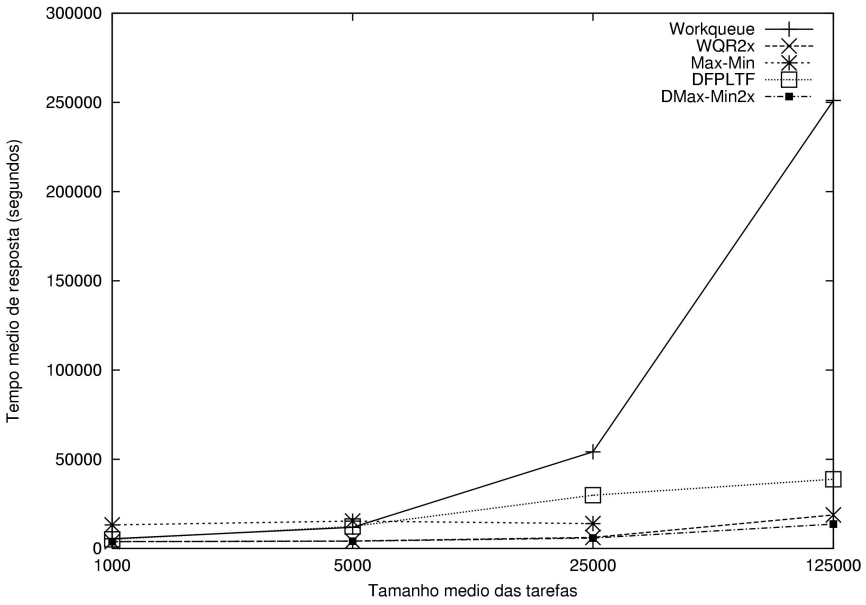

Figura 6.6: Desempenho das políticas de acordo com os diferentes tamanhos de tarefas sem considerar a política Max-Min.

vez, foge bastante desse comportamento por ser muito influenciada pela variação de carga do sistema, apresentando um desempenho inferior aos das outras políticas por ser altamente acoplada aos estados iniciais dos recursos do *grid*.

O comportamento nas políticas Workqueue e WQR2x explica-se pois a atribuição de uma tarefa pequena a uma máquina lenta no final da execução de uma aplicação não gera resultados tão catastróficos como no caso da atribuição de tarefas grandes, tal como é possível perceber nas tarefas com média de tamanho superiores a 25000 MI (Milhões de Instruções). Nesses casos, percebe-se que as políticas DFPLTF e DMax-Min2x apresentam melhores resultados pois utilizam informações do sistema para atribuir as tarefas às melhores máquinas disponíveis. As políticas Workqueue e WQR2x não contam com esse mecanismo, sendo que mesmo o processo da replicação de tarefas da WQR2x é incapaz de reverter essa perda no tempo de resposta.

A política DMax-Min2x apresentou resultados muito satisfatórios quando comparados aos de outras políticas. A tabela 6.3 apresenta o resumo dos ganhos de desempenho da DMax-Min2x em relação à Workqueue, WQR2x, Max-Min e DFPLTF.

A análise da tabela 6.3 mostra que a DMax-Min2x tende a ser superior às outras políticas de forma proporcional à granulosidade das tarefas submetidas ao sistema. Isso porque quanto maior o tamanho das tarefas, menor o número delas e, por conseqüência, maior o número de máquinas disponíveis para executá-las. Como a Max-Min utiliza conhec-

Tamanho médio das tarefas (MI)	1000	5000	25000	125000
Workqueue (%)	42	190	812	1931
WQR2x (%)	<1	3	6	44
Max-Min (%)	243	276	137	8131
DFPLTF (%)	32	201	407	184

Tabela 6.3: Média dos ganhos nos tempos de execução da política DMax-Min2x de acordo com os tamanhos médios de tarefas.

imentos do ambiente de escalonamento, ela é capaz de submeter as tarefas em espera aos processadores menos saturados.

O ganho de desempenho apresentado pela DMax-Min2x é especialmente importante quando ela é comparada às políticas Workqueue e DFPLTF. Primeiro porque a Workqueue consiste em uma política muito adotada em sistemas de *grid* e, segundo, porque a DMax-Min2x executa aplicações com uma média de tempo menor que a DFPLTF mesmo sem atualizar os estados dos recursos freqüentemente. Mesmo a vantagem da WQR2x em ser uma política que não necessita de informações a respeito das aplicações e nem dos recursos da plataforma, torna-se, de certa forma, um tanto quanto irrelevante devido à execução de tarefas com tempo de resposta significativamente menores com a utilização da DMax-Min2x.

Comportamento das Políticas com Diferentes Heterogeneidades de Recursos

A figura 6.7 ilustra o comportamento das políticas analisadas em *grids* compostos de recursos com diferenças em suas capacidades, sendo que cada ponto do gráfico representa a média das seis classes de tamanho de tarefas e seus respectivos 5 níveis de heterogeneidade.

Nos 5 níveis de heterogeneidade das máquinas, a política DMax-Min2x obteve os melhores resultados nos tempos de resposta das aplicações, apresentando um desempenho muito constante mesmo quando as diferenças entre as capacidades dos recursos se alterava. Comportamento semelhante a esse foi apresentando pelas políticas DFPLTF e WQR2x. Tal característica ocorre pois quanto maior o nível de heterogeneidade entre as máquinas do *grid*, maiores as chances de uma máquina mais lenta ser atribuída a uma tarefa menor que aquelas atribuídas aos recursos mais velozes, o que tende a balancear a carga entre essas máquinas.

Ao contrário das políticas DMax-Min2x, DFPLTF e WQR2x, bastante estáveis à diferença de capacidade entre as máquinas do *grid*, nas políticas Workqueue e Max-Min, observam-se mudanças nos seus desempenhos proporcionais ao aumento da heterogeneidade dos recursos. Na Workqueue essa mudança se reflete pelo aumento do tempo de resposta das aplicações. Tal fato acontece porque quando se aumenta a diferença entre as capacidades das

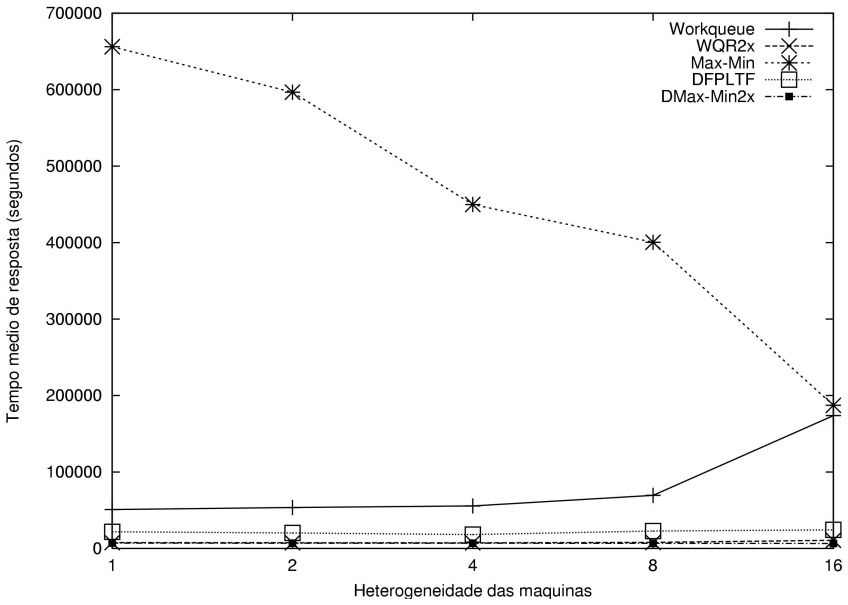

Figura 6.7: Desempenho das políticas de acordo com as diferentes heterogeneidades de máquinas.

máquinas, aumentam-se também as chances de ocorrer um desbalanceamento de carga no sistema, isto é, aumentam-se as chances de uma máquina lenta passar a ter mais probabilidade de receber uma tarefa que necessite de muito processamento. Por outro lado, como na Max-Min ocorre a coleta e análise de informações do sistema e das aplicações antes da submissão de tarefas, é possível distribuir a carga de trabalho de forma mais igualitária, atribuindo maiores tarefas aos processadores mais capazes, o que tende a resultar em melhores tempos de resposta com o aumento da heterogeneidade de recursos.

Comportamento das Políticas com Diferentes Heterogeneidades de Tarefas

Os experimentos realizados com as possíveis heterogeneidades de tarefas mostraram que essa é uma característica de pouca influência no tempo médio de resposta das aplicações. Os resultados obtidos nas simulações são ilustrados pela figura 6.8, onde cada ponto representa a média do tempo de execução das 4 classes de tamanho de tarefas e dos 5 níveis de heterogeneidade dos recursos.

Os resultados da política MaxMin, em especial, foram os únicos que apresentaram melhorias com o aumento da heterogeneidade de tarefas. Isso deve-se ao fato dessa heterogeneidade beneficiar aos recursos com maior capacidade do sistema, uma vez que na Max-Min, as tarefas com maior necessidade de processamento são atribuídas às máquinas com maior disponibilidade para processá-las.

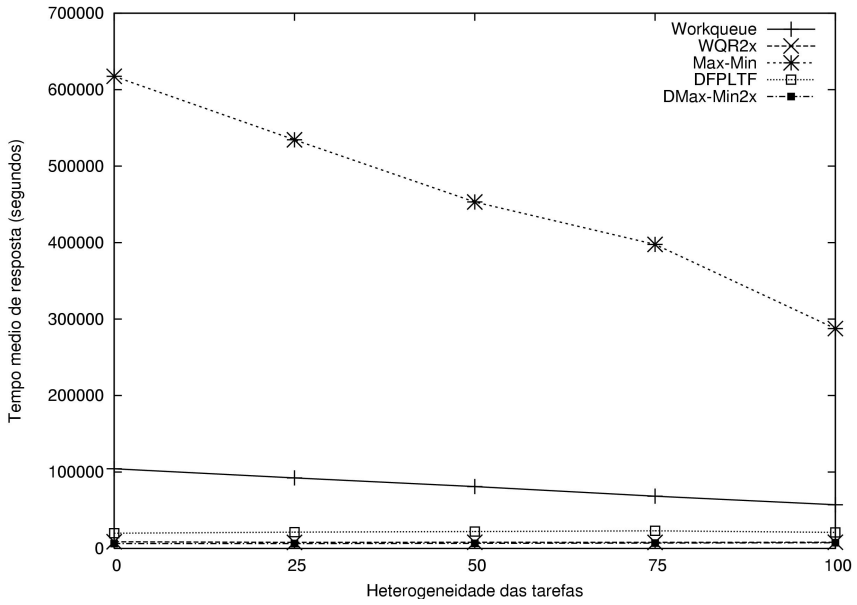

Figura 6.8: Desempenho das políticas de acordo com as diferentes heterogeneidades entre as tarefas de uma aplicação.

Estabilidade das Políticas Analisadas

A estabilidade das políticas foi estudada com base no número de repetição de experimentos necessários para se atingir um intervalo de confiança igual a 95%. Como nesse primeiro modelo de simulação as cargas dos recursos são orientadas a *traces* escolhidos aleatoriamente em cada iteração, o número de experimentos foi expressivamente mais alto em políticas que fazem uso de variáveis aleatórias, como no caso da Workqueue, por exemplo, e em políticas cujos tempos de resposta foram maiores, tais como a Max-Min e a DFPLTF.

De uma maneira geral, o número de iterações necessárias para se atingir a confiabilidade nas simulações foi bastante menor entre as políticas WQR2x e DMax-Min2x. Dessa forma nota-se a importância da replicação de tarefas com a finalidade de reduzir o tempo de resposta de aplicações e, por conseqüência, tornar o comportamento de políticas de escalonamento mais estáveis.

Além da aleatoriedade na escolha de qual *trace* utilizar na parametrização de carga de recursos, a granulosidade entre as tarefas de uma aplicação mostrou-se um parâmetro de bastante influência na estabilidade dos resultados das simulações, sendo que, políticas influenciadas de maneira negativa pelo tamanho das tarefas apresentaram uma maior instabilidade de comportamento, necessitando de um maior número de iterações para atingir a confiabilidade desejada. Tal instabilidade foi observada na política WQR2x que, apesar de continuar apresentando um número de iterações bem inferior aos das demais políticas (exceto

a DMax-Min2x), teve a sua estabilidade comprometida com o aumento da granulosidade das tarefas.

Uma Análise da Replicação de Tarefas

Outro ponto a considerar é o consumo de ciclos de CPU extras pela DMax-Min2x. Nas simulações notou-se que a média de consumo extra de processamento entre as políticas DMax-Min2x e WQR2x é semelhante. A tabela 6.4 expressa a média de desperdício de ciclos de CPU de tais políticas.

Tamanho Médio das Tarefas	1000	5000	25000	125000	Média
Média (%)	1	4	21	86	28

Tabela 6.4: Média de ciclos extras de CPU, em porcentagem, consumidos nas replicações das políticas DMax-Min2x e WQR2x.

A análise da tabela 6.4 mostra que o consumo de ciclos aumenta com o aumento do tamanho das tarefas, apresentando gastos insignificantes, menores que 1%, quando as tarefas possuem tamanho médio de 1000 MI até gastos bastante elevados, nos casos onde os experimentos tratam de tarefas de tamanho 125000 MI, onde o consumo extra de CPU gira em torno de 86%. Tal gasto, porém, muitas vezes torna-se aceitável em sistemas onde os recursos passam muito tempo ociosos tal como foi possível notar nos *traces* analisados, onde cerca de 50% das máquinas analisadas passam a maior parte do tempo sem realizar processamento.

6.5.2 Simulação Baseada em Modelos de Carga

A avaliação de políticas de escalonamento está especialmente ligada ao tipo de carga de trabalho que se utiliza. Dessa forma, é importante que se estabeleça uma carga de trabalho bem definida quando se deseja avaliar diferentes políticas de escalonamento. Alguns modelos para máquinas paralelas foram propostos, contudo, ainda não se conhece nenhum modelo de carga que represente o comportamento de aplicações em um *grid* computacional (FEITELSON & RUDOLPH, 1998; CALZAROSSA & GIUSEPPE, 1985; DOWNEY, 1998).

Neste segundo modelo de simulação, foi utilizado um modelo de carga de trabalho proposto para uso em clusters e máquinas paralelas, sendo que para a realização dos experimentos utilizou-se o schedSim[3], um simulador flexível de ambientes heterogêneos e implementado em Java. O schedSim simula o funcionamento de máquinas as quais executam seus

[3]O código fonte do schedSim pode ser encontrado em http://www.icmc.usp.br/ mello/outr.html.

processos de forma cíclica de acordo com um *quantum* de tempo, tal como ocorre no Sistema Operacional Linux.

Simulação das Aplicações

Por falta de modelos de carga de trabalho orientados a *grids* computacionais, as aplicações utilizadas na simulação consistem de aplicações paralelas, onde as tarefas pertencentes a uma mesma aplicação, possuem um mesmo número de instruções, tal qual no modelo proposto por Feitelson (FEITELSON & RUDOLPH, 1998; LO *et al.*, 1998; MU'ALEM & FEITELSON, 2001).

Assim como na simulação descrita na seção 6.5.1, diferentes cenários são estudados com o objetivo de analisar o comportamento da DMax-Min2x sob diferentes aspectos, seja quando há pouca carga para muitos computadores ou, quando há muita carga para poucos computadores. Dessa forma, foram simulados 5 cenários para as aplicações onde o número de tarefas em cada uma delas pode chegar a até 8, 32, 64, 128 ou 256 tarefas. Apesar de cada tarefa pertencente a uma mesma aplicação possuir a mesma quantidade de processamento, a quantidade de memória estática necessária para a execução de cada uma delas segue uma função de distribuição de probabilidades exponencial de média 300 Kbytes.

Simulação dos Recursos

A parametrização dos recursos da simulação foi realizada de acordo com dados colhidos em experimentos reais em máquinas de um laboratório de pesquisa utilizando-se do *benchmark* proposto por Mello e Senger[4].

O ambiente simulado consiste em 32 máquinas nas quais a capacidade de processamento, a memória principal e a memória virtual são definidas através de funções de distribuição de probabilidades com médias iguais a 1500 Mips para o processamento e 1024 Mbytes para as memórias.

Simulação das Cargas de Trabalho dos Recursos

No modelo proposto por Feitelson, utiliza-se uma taxa de chegada de processos orientada por uma função de distribuição de probabilidades exponencial de média 1500 segundos. Por acreditar-se que em um *grid* a quantidade de trabalho que chega ao ambiente é maior,

[4]Disponível em http://www.icmc.usp.br/ mello.

considerou-se, então, uma taxa de chegada segundo uma função de distribuição de probabilidades exponencial de média 100 segundos.

A adoção de uma taxa de chegada menor é capaz de representar com maior fidelidade o comportamento dinâmico do *grid* onde a carga dos processadores varia bastante com o tempo. Neste modelo, considera-se um sistema aberto, onde novas aplicações podem chegar a qualquer momento e disputarão, com os processos já em execução, por um *quantum* do processador, ao contrário do que ocorre na simulação anterior, onde somente uma aplicação por vez do *grid* entra em execução juntamente com os processos pertencentes ao usuário da máquina executora.

Nas simulações, a variação de carga de trabalho foi representada não somente pela redução da taxa de chegada de processos e pelo número máximo de tarefas que cada tipo de aplicação pode ter, como também pelo número de aplicações que chegam ao sistema. Dessa forma, para cada tipo de aplicação foram executadas simulações com 1, 10, 20, 30, 40, 50, 100, 150, 200 e 250 aplicações cada uma. Considerando que foram realizadas simulações com 5 tipos de aplicações (de até 8, 32, 64, 128 e 256 tarefas), totalizam-se 50 tipos de cenários simulados.

Análise dos Resultados

A análise dos resultados sob diferentes níveis de carga mostrou que a DMax-Min2x obtém melhores resultados quando a variação de carga de trabalho no sistema é grande. Dessa forma, as simulações possuindo maiores números de tarefas por aplicação são as que apresentam maiores ganhos de desempenho. Para uma melhor visualização dos resultados gerados, são apresentados dois gráficos para cada tipo de aplicação simulada. No primeiro gráfico, são expostos os resultados obtidos em cada uma das políticas, a citar a Workqueue, a WQR2x, a Max-Min, a DFPLTF e a DMax-Min2x. Como as políticas Workqueue e WQR2x tendem a alcançar tempos de resposta maiores que as outras políticas, elas foram excluídas do segundo gráfico para que se pudesse visualizar com maior precisão os resultados obtidos pela Max-Min, pela DFPLTF e pela DMax-Min2x. É importante lembrar que cada ponto nos gráficos representa a média obtida nas simulações até que um intervalo de confiança de 95% fosse obtido.

As figuras 6.9 e 6.10 ilustram os resultados da simulação com aplicações possuindo até 8 tarefas cada uma. É possível perceber uma similaridade de desempenho entre as políticas Max-Min, DFPLTF e DMax-Min2x. Isso justifica-se pois há pouca carga de trabalho no sistema considerando que muitas aplicações possuem poucas tarefas (2 ou 3). Como não há

muita variação de carga, a política Max-Min é beneficiada e atinge desempenho semelhante à DFPLTF a qual trata-se de uma política dinâmica. Pelo gráfico 6.9, ainda é possível perceber a grande vantagem em se utilizar replicação na política Workqueue, sendo que nesse cenário de simulação o uso de réplicas pela WQR2x melhorou bastante os resultados obtidos pela Workqueue. Tal ganho, porém, foi insuficiente para aproximar de forma significativa os resultados obtidos pela WQR2x em relação às demais políticas.

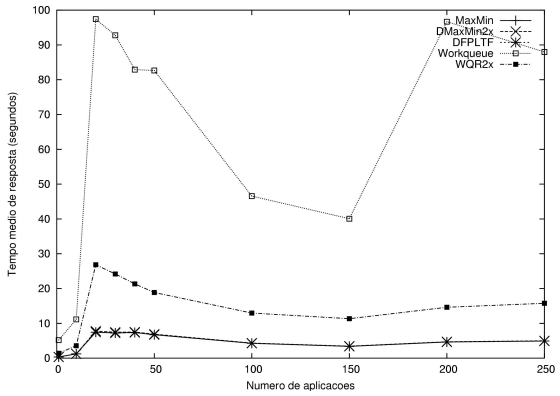

Figura 6.9: Comportamento das políticas de escalonamento simuladas com aplicações de até 8 tarefas.

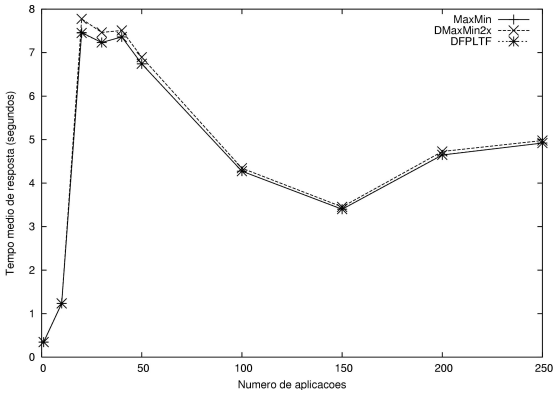

Figura 6.10: Comportamento das políticas Max-Min, DFPLTF e DMax-Min2x com aplicações de até 8 tarefas.

A utilização de aplicações com um número maior que 8 tarefas por aplicação é capaz de alterar o comportamento das políticas analisadas. Na simulação com aplicações com até 32 tarefas cada uma, a política DMax-Min2x começa a apresentar tempos de resposta menores que as demais políticas assim como ilustram as figuras 6.11 e 6.12. Esse ganho de desempenho, apesar de pequeno, já expressa a interferência da variação de carga na política Max-Min que, por ser estática, é incapaz de se adequar à mudanças do sistema.

O ganho de desempenho da DMax-Min2x começa a ser bastante significativo quando são simuladas aplicações onde o número de tarefas pode chegar a 64. As figuras 6.13 e 6.14[5] ilustram o comportamento da política analisada cujo tempo de resposta gira em torno de 35% menor que a segunda melhor política: a DFPLTF. Através dos gráficos percebe-se uma tendência das políticas Max-Min, DFPLTF e DMax-Min2x apresentarem resultados semelhantes que, ao aumentar o número de aplicações nas simulações, se diferenciam mais devido à maior carga de trabalho imposta ao sistema.

Se por um lado, o maior número de tarefas no sistema proporciona piores resultados à política Max-Min, por outro, ele melhora o desempenho obtido pela WQR2x. Isso porque

[5]Nesses gráficos, os resultados gerados pela execução de até 250 aplicações foram ocultados por apresentarem números muito grandes e, assim, dificultarem a qualidade de visualização das figuras.

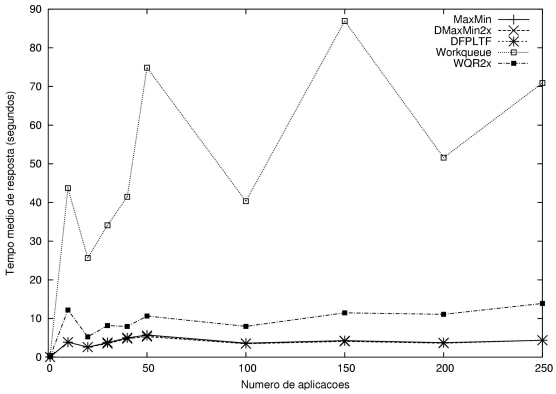

Figura 6.11: Comportamento das políticas de escalonamento simuladas com aplicações de até 32 tarefas.

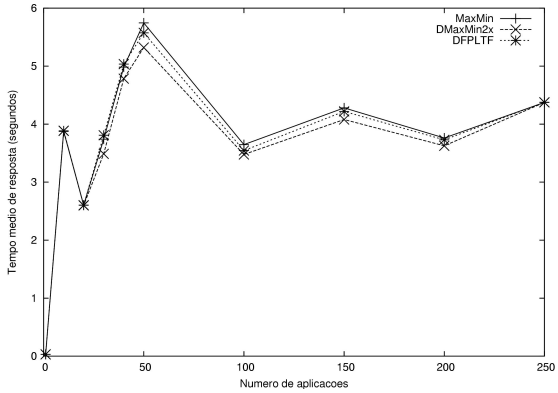

Figura 6.12: Comportamento das políticas Max-Min, DFPLTF e DMax-Min2x com aplicações de até 32 tarefas.

com um maior número de tarefas por aplicação, a distribuição de carga entre os recursos é mais balanceada, pois enquanto uma máquina lenta estiver processando uma grande tarefa, as demais podem processar um número maior de pequenas tarefas, o que tende a resultar em um tempo de resposta similar entre os recursos.

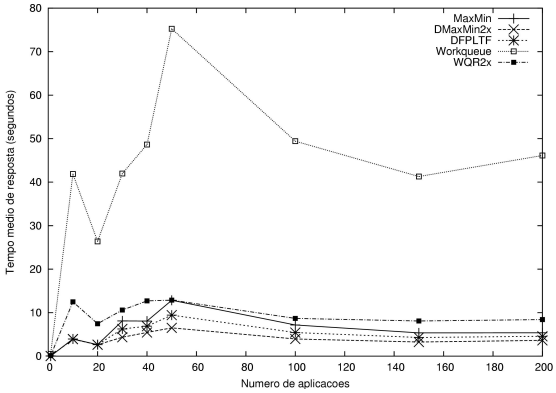

Figura 6.13: Comportamento das políticas de escalonamento simuladas com aplicações de até 64 tarefas.

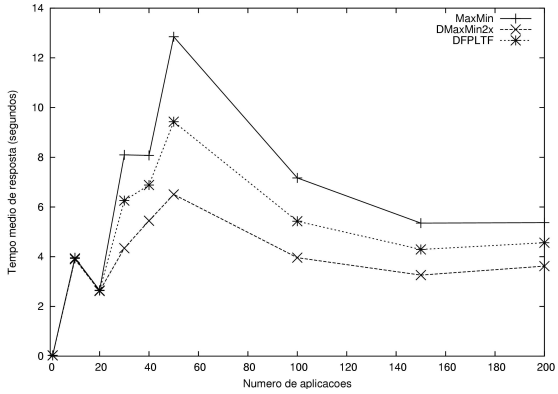

Figura 6.14: Comportamento das políticas Max-Min, DFPLTF e DMax-Min2x com aplicações de até 64 tarefas.

Quando aplicações de até 128 tarefas foram simuladas a política Max-Min passou a apresentar um desempenho inferior à todas as demais políticas, exceto a Worqueue, enquanto que a WQR2x obteve resultados semelhantes à DFPLTF. Esse fato, expresso nas figuras 6.15 e 6.16, mostra a importância da adoção de políticas dinâmicas no *grid*, as quais mesmo utilizando-se de mecanismos bastante simples, são capazes de se adequar à mudanças do sistema e, dessa forma, atingir melhores resultados que políticas mais elaboradas, tais como a Max-Min, porém, que realizam o escalonamento de forma estática.

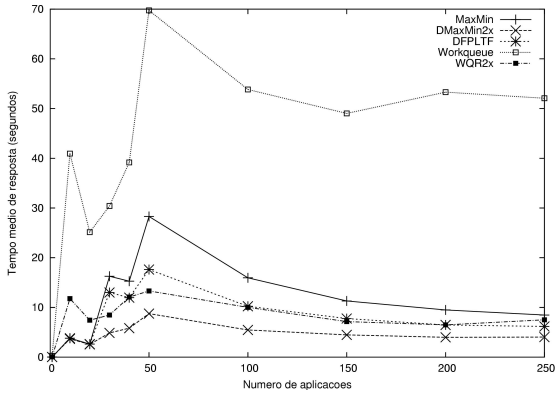

Figura 6.15: Comportamento das políticas de escalonamento simuladas com aplicações de até 128 tarefas.

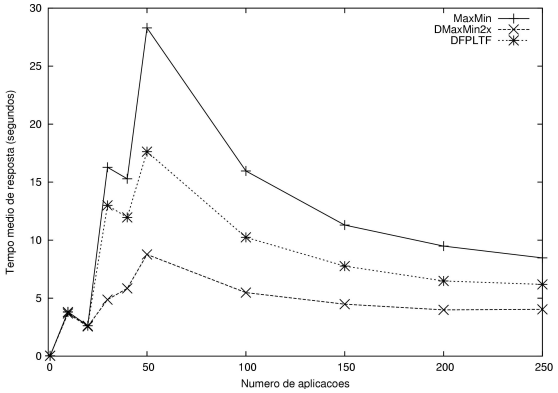

Figura 6.16: Comportamento das políticas Max-Min, DFPLTF e DMax-Min2x com aplicações de até 128 tarefas.

Por fim, os resultados das simulações com aplicações de até 256 tarefas são apresentados nas figuras 6.17 e 6.18. Mais uma vez, o desempenho da DMax-Min2x foi superior aos das outras políticas. A política Max-Min manteve a tendência de degradação de desempenho e apresentou resultados melhores apenas que a Workqueue. Enquanto isso, a WQR2x, quando comparada à Workqueue, apresentou uma melhora, em média, de 4 vezes no tempo de resposta através da utilização de replicações. O mesmo uso de réplicas aliado ao escalonamento dinâmico, proposto pela DMax-Min2x, garantiu a essa política, um ganho de desempenho de 43% quando comparada à DFPLTF e 176% quando comparada à Max-Min. Dessa forma, percebe-se que a replicação de tarefas é um artifício que pode ser utilizado mesmo em ambientes onde novas aplicações chegam a qualquer momento, sendo que a duplicação de tarefas não chega a degradar a execução de outros processos do sistema.

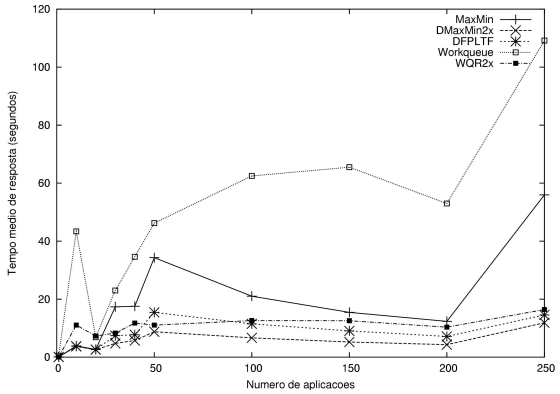

Figura 6.17: Comportamento das políticas de escalonamento simuladas com aplicações de até 256 tarefas.

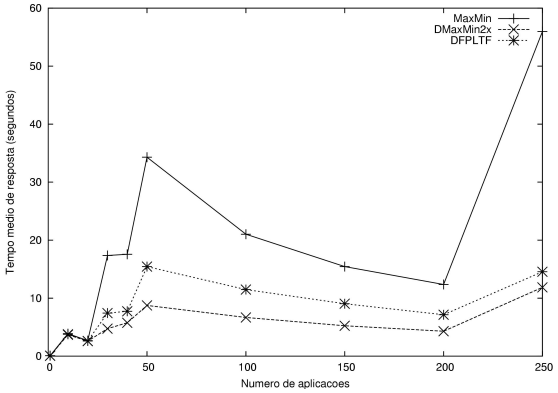

Figura 6.18: Comportamento das políticas Max-Min, DFPLTF e DMax-Min2x com aplicações de até 256 tarefas.

Como no modelo de simulação anterior, a política DMax-Min2x se apresentou bastante estável, necessitando de um número pequeno de iterações (em média 2) para atingir a confiabilidade desejada, assim como as políticas Max-Min e DFPLTF que apresentaram comportamento semelhante. As políticas Workqueue e WQR2x por atribuírem tarefas de forma aleatória mostraram uma instabilidade inversamente proporcional à quantidade de processos no sistema, isto é, quanto menor a relação tarefas por máquinas, menor tende a ser o desempenho dessas políticas e maior as suas instabilidades.

6.6 Considerações Finais

Grids computacionais têm emergido no cenário de Computação Paralela Distribuída por representar uma opção para o processamento de tarefas a um baixo custo. Visando a popularização desse tipo de plataforma, o projeto do Globus tem se destacado por apresentar soluções para a criação de infra-estruturas e o desenvolvimento de serviços em *grid*. Porém, um dos gargalos presentes em sistemas de *grid* e que ainda se encontra de forma ineficaz no Globus, trata-se do escalonamento de processos, visto que, em um *grid*, a submissão de tarefas deve ser realizada de forma a atribuir tarefas aos processadores menos saturados, visando balancear a carga entre os recursos. Sendo assim, neste capítulo, foi proposta a construção de um *framework* que, através da utilização dos serviços disponibilizados pelo Globus, tais como a autenticação, com uso de chaves criptografadas, e a transferência de dados, por meio da inicialização de servidores HTTP, realiza a atribuição de tarefas aos recursos presentes no sistema de forma pré-determinada através de políticas de escalonamento inseridas pelo usuário que submete a aplicação. Dessa forma, pode-se dizer que o escalonamento no *framework* proposto é orientado à aplicação e não ao sistema, uma vez que, ele tende a satisfazer aos interesses de um usuário em particular.

Além da construção de uma infra-estrutura para escalonamento de processos em um nível global, chamado de meta-escalonamento, este capítulo propôs a adoção de um ambiente de escalonamento dinâmico e flexível, o AMIGO, para ser empregado no gerenciamento da atribuição de processos em grupos com um número de recursos mais reduzido, os quais executam processos dos ambientes de passagem de mensagem MPI e PVM. Tais ambientes de passagem de mensagem têm o seu código alterado para chamarem às rotinas do AMIGO sempre que uma nova atribuição de tarefas deve ser realizada, sendo que, assim como no *framework* proposto, o escalonamento é realizado de acordo com políticas já inseridas no ambiente. A adoção do AMIGO como um ambiente de escalonamento local é especialmente importante pois ele é capaz de trabalhar com aplicações cujas tarefas realizam comunicação

entre si. Apesar dessas aplicações serem menos indicadas para a execução em plataformas altamente distribuídas, existem casos onde a comunicação entre tarefas é mínima e imprescindível, sendo que tais casos podem ser beneficiados se possuírem códigos compatíveis com as versões do MPI e do PVM que foram alteradas.

A adoção de um ambiente de escalonamento flexível sugere a importância de se utilizar políticas de escalonamento orientadas ao tipo de aplicação que se deve tratar. Dessa forma, neste capítulo foi proposta uma nova política de escalonamento intitulada *Dynamic* Max-Min2x. Tal política utiliza-se de escalonamento dinâmico e de replicação de tarefas para a redução do tempo de resposta de aplicações do tipo *Bag-of-Tasks*, isto é, aplicações que não realizam comunicação entre suas tarefas. Através de simulações, foi mostrado que a DMax-Min2x supera, em muitos cenários, outras políticas antes propostas para o escalonamento de aplicações em *grid*, tais como a Workqueue, a WQR2x, a Max-Min e a DFPLTF.

No primeiro modelo de simulação utilizado, a carga dos recursos é orientada por *traces* com a utilização de CPU de cada máquina em função do tempo. Trata-se portanto de um sistema onde somente é possível a chegada de processos do usuário da máquina local e de uma tarefa do *grid* por vez, sendo que não é possível, neste modelo, avaliar a interferência que a execução de processos externos causam nos processos locais. Por outro lado, é possível estimar através da porcentagem de processamento disponível, o quanto de ciclos extras são necessários para a adoção de réplicas na WQR2x e na DMax-Min2x.

No segundo modelo de simulação analisado, o sistema comporta-se de maneira aberta, isto é, é possível a chegada de diversos processos distribuídos de acordo com uma função de distribuição de probabilidades exponencial. Nesse modelo, todos os processos atribuídos às máquinas são executados cada um em um *quantum* de tempo de maneira cíclica, assim como acontece nos sistemas Linux. Dessa forma, é possível observar a interferência que a chegada de um número grande de processos ou mesmo a replicação de tarefas pode causar no sistema. A parametrização dos recursos e das aplicações nesse modelo são feitas baseadas em dados reais e, por esse motivo, tal modelo tende a se aproximar bastante de ambientes reais.

Pode-se dizer que os dois modelos de simulação utilizados, abrangem paradigmas diferentes de computação em *grid*. O primeiro é mais adequado à avaliação de políticas para infra-estruturas de *grid* formadas por computadores pessoais em estado ocioso ligados em rede mundial. Um exemplo desse paradigma ocorre no projeto SetAtHome, onde somente uma tarefa da aplicação como um todo é submetida à uma determinada máquina por vez, competindo apenas com os processos locais do donatário por recursos de processamento. O segundo modelo representa de maneira mais adequada *grids* formados por computadores

com maior capacidade, tais como máquinas paralelas e *clusters*, os quais, geralmente, são formados por um conjunto de usuários que submetem concorrentemente os seus processos para execução no sistema.

Em ambos os modelos de simulação foi possível notar as vantagens em se utilizar a DMax-Min2x como uma política de escalonamento em sistemas de *grid*, principalmente quando há variação de carga nos recursos, característica comumente encontrada em *grids* computacionais. Por essa variação de carga de trabalho presente nas simulações, ficou evidenciado a necessidade da utilização de políticas dinâmicas ao invés de políticas estáticas incapazes de reagir à mudanças do sistema. A Max-Min, por exemplo, em muitos casos estudados, apresentou desempenho inferior à Workqueue, uma política simples e que não necessita de nenhuma informação a respeito dos recursos e das aplicações, tal como ocorre na Max-Min.

O próximo capítulo, apresenta as conclusões gerais obtidas nesta dissertação, assim como detalha as contribuições geradas no desenvolvimento do projeto e enumera algumas propostas para continuidade deste trabalho.

Capítulo 7

Conclusão

7.1 Conclusões Gerais

Esta dissertação mostrou a importância da utilização de *grids* computacionais como uma plataforma de execução para a realização de Computação Paralela Distribuída. A revisão bibliográfica apresentada nos Capítulos 2 e 3 apresenta conceitos, definições e tendências que fazem dos sistemas em *grid* uma crescente área de pesquisa no meio computacional e que, pelo curto período de estudos, ainda apresenta muitas controvérsias e mitos a seu respeito.

Como uma forma de estudo de caso e exemplificação das funcionalidades básicas que um *grid* deve possuir, o Capítulo 4 descreve os serviços presentes no conjunto de ferramentas Globus, um sistema para computação em *grid* largamente difundido no momento, o qual possibilita a criação e utilização de infra-estruturas em *grid*. Uma análise mais focada no gerenciamento de processos do Globus, porém, mostra que esse é ainda um serviço que necessita de novas funcionalidades, principalmente, quando se trata de formas para a atribuição de tarefas no sistema. Sendo assim, no Capítulo 5, os problemas relacionados ao escalonamento de processos em tal plataforma são evidenciados e discutidos com a descrição de algumas métricas de desempenho e políticas propostas para o gerenciamento dos recursos disponíveis do *grid*.

Feita a exposição das dificuldades encontradas no escalonamento de processos em *grid*, e dadas as características bastante peculiares desses sistemas, faz-se necessário o desenvolvimento de novas formas de escalonamento de tarefas. Sendo assim, o Capítulo 6 propõe

meios para se atribuir tarefas em um *grid* com a utilização dos serviços do Globus e com o apoio do AMIGO, um ambiente de escalonamento flexível e dinâmico. Trata-se da construção de um *framework* onde as tarefas de uma aplicação, antes de serem submetidas aos recursos do *grid*, são escalonadas de acordo com políticas de escalonamento inseridas pelo cliente. Essa abordagem permite que cada usuário identifique as suas necessidades de serviço e expresse-as através de regras que aplicadas pelo *framework* geram resultados orientados à satisfação do usuário.

A análise das políticas encontradas na literatura para o escalonamento de aplicações que não realizam comunicação entre suas tarefas, culminou na proposta de uma nova política de escalonamento baseada nas principais características das políticas analisadas tendo como objetivo a redução do tempo médio de resposta das aplicações. Para isso, a proposta da política, denominada *Dynamic* Max-Min2x, submete as maiores tarefas de uma aplicação aos processadores do sistema que apresentam os melhores tempos de resposta para elas. Esse procedimento visa o balanceamento de carga, mas não evita que um recurso com pouca capacidade receba uma tarefa que necessite de muito processamento. Para contornar esse problema, evitando que o tempo de resposta das aplicações torne-se alto, a política proposta utiliza duas outras técnicas para a atenuação da heterogeneidade e da variação de carga no *grid*: a atribuição dinâmica e a replicação de tarefas.

Na atribuição dinâmica de tarefas, quando um recurso termina a execução de uma tarefa, ele recebe outra pertencente à mesma aplicação e que se encontra em espera por execução. Quando não houver mais tarefas em espera, os processadores, à medida que terminam seus processos, começam a replicar tarefas de outras máquinas. Quando uma das réplicas termina a execução, as demais são canceladas. Como na *Dynamic* Max-Min2x cada tarefa pode ser replicada apenas uma vez, ela recebe a extensão 2x.

A avaliação da DMax-Min2x foi feita através de simulações com a utilização de dois modelos encontrados na literatura os quais possuem abordagens distintas de computação em *grid*. Ambos apontaram a DMax-Min2x como uma política bastante estável que, apesar de sua simplicidade, apresenta resultados melhores que os das políticas Workqueue, WQR2x, Max-Min e DFPLTF. Análises dos resultados da DMax-Min2x mostram que ela tende a ser superior às demais políticas, principalmente, em cenários onde há variação na carga de trabalho dos processadores e as aplicações necessitam de maior quantidade de processamento.

Os resultados obtidos nos experimentos da *Dynamic* Max-Min2x mostram que a adoção de regras simples no escalonamento de tarefas pode apresentar ganhos de desempenho significativos no tempo de resposta de aplicações no *grid*. Dessa forma, espera-se que este

trabalho contribua na utilização e na adoção de políticas mais orientadas às características de sistemas tais como *grids* e, conseqüentemente, na maior aceitação de tais sistemas para a execução de aplicações paralelas.

7.2 Contribuições deste Trabalho

O desenvolvimento desta dissertação abordou diversos aspectos presentes na computação em *grid*, sendo que as principais contribuições são:

- Uma análise crítica da bibliografia existente a respeito de *grids* computacionais, enfocando as principais dúvidas a respeito da definição e utilização dos sistemas em *grid*;
- O estabelecimento de uma nova linha de pesquisa do grupo de Sistemas Distribuídos e Programação Concorrente do ICMC/USP, com a expectativa de que novos trabalhos em *grid* sejam desenvolvidos;
- A descrição das principais características presentes em *grids*, muitas vezes, inexistentes em outras plataformas computacionais e a apresentação das principais dificuldades e limitações acarretadas por tais características;
- O estudo das funcionalidades presentes no conjunto de ferramentas Globus, como tais serviços foram implementados e de que forma eles podem ser utilizados na construção de novas aplicações que prestem serviços diferenciados;
- O projeto de um *framework* flexível para a submissão de tarefas em um *grid* computacional, o qual pode ser instalado e configurado em computadores para a construção de uma infra-estrutura de *grid*, onde experimentos podem ser avaliados de maneira real;
- A identificação dos principais problemas encontrados no escalonamento de processos em *grids* computacionais e quais as características que devem estar presentes em políticas de escalonamento orientadas para o uso em tais plataformas;
- A integração entre o Globus e o ambiente de escalonamento AMIGO, desenvolvido em um trabalho de doutorado, o que reforça a integração dos trabalhos de pesquisa realizados no grupo de Sistemas Distribuídos e Programação Concorrente do ICMC/USP;
- A análise de duas ferramentas para simulação de ambientes de *grid* juntamente com dois modelos para a parametrização de cargas e recursos. A implementação de tais modelos

em programas de simulação com as diferentes políticas de escalonamento estudadas poderão ser utilizados em outros trabalhos;

- A proposta de uma política de escalonamento orientada à redução do tempo de resposta de aplicações do tipo *Bag-of-Tasks*. Tal política pode ser adotada no gerenciamento de recurso de sistemas reais de *grid*.

7.3 Trabalhos Futuros

Devido ao fato de *grids* computacionais representarem uma área de pesquisa bastante recente e dado a abrangência e importância que escalonamento de processos têm em tais plataformas, pode-se citar alguns trabalhos que vêm a contribuir para a expansão dos resultados obtidos neste projeto, tais como:

- A proposta de um modelo de carga que represente de maneira fiel o perfil de aplicações que, freqüentemente, são submetidas para execução em *grids* computacionais;
- A proposta de um modelo, possivelmente baseado em *logs*, que expresse a taxa de chegada de processos em um *grid*;
- A construção de uma infra-estrutura para o aproveitamento de recursos em estado ocioso, em instituições tais como universidades. A capacidade de processamento que se encontrar sendo desperdiçada pode ser utilizada para a execução de aplicações do tipo *Bag-of-Tasks*, por exemplo;
- A proposta de um novo sistema para construção de *grids*, mas que seja implementado com a utilização de técnicas mais simples que as adotadas pelo Globus;
- O estudo de novas políticas de escalonamento onde as aplicações realizem pequenas comunicações entre as suas tarefas;
- O estudo da inclusão de migração de processos na política DMax-Min em vez da utilização de replicação de tarefas;
- A validação da política DMax-Min2x em ambientes de *grid* reais com aplicações também reais.

Referências Bibliográficas

ALMASI, G. S.; GOTTLIEB, A. (1994). *Highly Parallel Computing*. Cummings, 2 edição.

AMORIM, C. L.; BARBOSA, V. C.; FERNANDES, E. S. T. (1988). *Uma Introdução à Computação Paralela e Distribuída*. Unicamp, Campinas, Brasil.

ARAUJO, A. P. F. (1999). DPWP: Uma Nova Abordagem ao Escalonamento Dinâmico em Computação Paralela Virtual. Dissertação (Mestrado), ICMC-USP, São Carlos, Brasil.

BERMAN, F. (1998). *The Grid: Blueprint for a New Computing Infrastructure*, chapter High-performance schedulers, p. 279–309. Morgan Kaufmann, San Francisco, USA.

BERSTIS, V. (2002). Fundamentals of Grid Computing. Relatório técnico, IBM. Disponível em http://www.redbooks.ibm.com/redpapers/pdfs/redp3613.pdf.

BUYYA, R. (1999). *High Performance: Cluster Computing*, v. 2. Prenctice Hall.

BUYYA, R.; ABRAMSON, D.; GIDDY, J. (2001). A Case for Economy Grid Architecture for Service-Oriented Grid Computing. *10th IEEE International Heterogeneous Computing Workshop (HCW 2001), In conjunction with IPDPS 2001*, p. 83–83, San Francisco, USA.

BUYYA, R.; KRAUTER, K.; MAHESWARAN, M. (2002). A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. *Software - Practice and Experience*, v.32, n.2, p.135–164.

CALZAROSSA, M.; GIUSEPPE, S. (1985). A characterization of the variation in time of workload arrival patterns. *IEEE Transactions on Computers*, v. C-34, p. 156–162.

CASANOVA, H.; LEGRAND, A.; ZAGORODNOV, D.; BERMAN, F. (2000). Heuristics for scheduling parameter sweep applications in grid environments. *9th Heterogeneous Computing Workshop (HCW)*, p. 349–363, Cancun, Mexico.

CASAVANT, T. L.; KUHL, J. G. (1988). A Taxonomy of Scheduling in General-purpose Distributed Computing Systems. *IEEE Transactions on Software Engineering*, v.14, n.2, p.141–154.

CIRNE, W. (2002). Computational grids: Architectures, tecnologies and applications. *Terceiro Workshop em Sistemas Computacionais de Alto Desempenho*, Vitória, Brasil.

COG (2005). Página dos Commodity Grid (CoG) Kits. Disponível em http://www.cogkit.org/.

CONDOR (2005a). Página do projeto Condor. Disponível em http://www.cs.wisc.edu/condor/.

CONDOR (2005b). Página do projeto Condor-G. Disponível em http://www.cs.wisc.edu/condor/condorg/.

COULOURIS, G. F.; DOLLIMORE, J.; JINDBERG, T. (2001). *Distributed systems: Concepts and Design*. Addison-Wesley, 3 edição.

CZAJKOWSKI, K.; FOSTER, I.; KARONIS, N.; KESSELMAN, C.; STUART, M.; SMITH, W.; TUECKE, S. (1998). A resource management architecture for metacomputing systems. *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, p. 62–82. Springer-Verlag LNCS 1459.

DOWNEY, A. B. (1998). A parallel workload model and its implications for processor allocation. *Cluster Computing*, v.1, n.1, p.133–145.

ENTROPIA (2005). Página da companhia Entropia - PC Grid Solutions. Disponível em http://www.entropia.com.

FEITELSON, D. G.; RUDOLPH, L. (1998). Metrics and benchmarking for parallel job scheduling. *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, v. 1459, p. 1–24, London, UK. Springer-Verlag.

FERREIRA, L.; BERSTIS, V.; ARMSTRONG, J.; KENDZIERSKI, M.; NEUKOETTER, A.; TAKAGI, M.; BING-WO, R.; ADEEB, A.; MURAKAWA, R.; HERNANDEZ, O.; MAGOWAN, J.; BIEBERSTEIN, N. (2003a). *Introduction to Grid Computing with Globus*. IBM's International Technical Support Organization.

FERREIRA, L.; JACOB, B.; S., S.; BROWN, M.; SUNDARARAJAN, S.; LEPESANT, J.; BANCK, J. (2003b). *Globus Toolkit 3 Early Experiences*. IBM's International Technical Support Organization.

FERREIRA, L.; THAKORE, A.; BROWN, M.; LUCCHESE, F.; RUOBO, H.; LIN, L.; MANESCO, P.; MAUSOLF, J.; MOMTAHENI, N.; SUBBIAN, K.; HERNANDEZ, O. (2004). *Grid Services Programming and Application Enablement*. IBM's International Technical Support Organization.

FLYNN, M. J. (1972). Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, v.21, n.9.

FOSTER, I. (2002). What is the Grid? A Three Point Checklist. Relatório técnico, Argonne National Laboratory and University of Chicago. Disponível em http://www-fp.mcs.anl.gov/ foster/Articles/WhatIsTheGrid.pdf.

FOSTER, I.; KESSELMAN, C. (1999). *The Grid: Blueprint for a New Computing Infrastructure*, chapter Computational Grids. Morgan-Kaufman.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Lecture Notes in Computer Science*, v.2150.

GGF (2005). Página do Global Grid Forum. Disponível em http://www.ggf.org/.

GLOBUS (2005). Página do projeto Globus. Disponível em http://www.globus.org.

GOLDCHLEGER, A.; KON, F.; LEJBMAN, A. G.; FINGER, M.; SONG, S. W. (2002). InteGrade: Rumo a um Sistema de Computação em Grade para Aproveitamento de Recursos Ociosos em Máquinas Compartilhadas. Relatório técnico, MAC-IME-USP. Disponível em http://gsd.ime.usp.br/publications/rt-integrade.pdf.

JACOB, B.; FERREIRA, L.; BLEBERSTEIN, N.; GILZEAN, C.; GIRARD, J.; STRACHOWSKI, R.; YU, S. (2003). *Enabling Applications for Grid Computing with Globus*. IBM's International Technical Support Organization.

JAIN, H. (1991). *The Art of Computer Systems Performance Analysis: Tecniques for Experimental Design Measurement, Simulation, and Modeling*. Wiley.

JAMES, A. H.; K., H. A.; CODDINGTON, D. P. (1999). Scheduling independent tasks on metacomputing systems. *Parallel and Distributed Computing Systems (PDCS'99)*, p. 156–162, Fort Lauderdale, Florida, USA.

JAMES, F.; TODD, T.; FOSTER, I.; LIVNY, M.; TUECKE, S. (2002). Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, v.5, p.237–246.

LASZEWSKI, G.; ALUNKAL, B.; AMIN, K.; GAWOR, J.; HATEGAN, M.; NIJSURE, S. (2004). *The Java CoG Kit User Manual - Draft Version 1.1.* Manual disponível em http://www.globus.org/cog/manual-user.pdf.

LASZEWSKI, G.; FOSTER, I.; GAWOR, J.; SMITH, W.; TUECKE, S. (2000). CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. *ACM Java Grande 2000 Conference*, p. 97–106, San Francisco, CA.

LAW, A. M.; KELTON, D. W. (1982). *Simulation Modeling and Analysis.* McGraw-Hill series in industrial engineering and management science. McGraw-Hill, New York, USA, 2 ed. edição.

LEGION (2005). Página do projeto Legion. Disponível em http://www.legion.virginia.edu.

LO, V.; MACHE, J.; WINDISCH, K. (1998). A comparative study of real workload traces and synthetic workload models for parallel job scheduling. Feitelson, D. G.; Rudolph, L., editores, *Job Scheduling Strategies for Parallel Processing*, p. 25–46. Springer Verlag. Lect. Notes Comput. Sci. vol. 1459.

LOAD LEVELER (2005). Página do sistema de escalonamento LoadLeveler. Disponível em http://www.mhpcc.edu/training/workshop/loadleveler/MAIN.html.

LSF (2005). Página do sistema de escalonamento Load Sharing Facility (LSF). Disponível em http://accl.grc.nasa.gov/lsf/.

MAHESWARAN, M.; ALI, S.; SIEGEL, H. J.; HENSGEN, D. A.; FREUND, R. F. (1999). Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. *8th Heterogeneous Computing Workshop*, p. 30–44.

MELLO, R. F.; SENGER, L. (2004). A new migration model based on the evaluation of processes load and lifetime on heterogeneous computing environments. *SBAC-PAD - 16th Symposium on Computer Architecture and High Performance Computing*, Foz do Iguaçu, PR, Brasil.

MU'ALEM, A. W.; FEITELSON, D. G. (2001). Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. *IEEE Trans. Parallel Distrib. Syst.*, v.12, n.6, p.529–543.

NWS (2005). Página do sistema Network Weather Service (NWS). Disponível em http://nws.cs.ucsb.edu/.

OURGRID (2005). Página do projeto OurGrid. Disponível em http://www.ourgrid.com.

PARABON (2005). Página do projeto Pioneer. Disponível em http://www.parabon.com.

PBS (2005). Página do sistema de escalonamento Portable Batch System (PBS). Disponível em http://www.openpbs.org/.

SENGER, L. J.; MELLO, R. F.; SANTANA, M. J.; SANTANA, R. C. (2004). An On-line Approach for Classifying and Extracting Application Behavior on Linux. *High Performance Computer: Paradigm and Infrastructure*. John Wiley and Sons Inc.

SETIATHOME (2005). Página do projeto SetiAtHome. Disponível em http://setiathome.ssl.berkeley.edu.

SHIVARATRI, N. G.; KRUEGER, P.; SINGHAL, M. (1992). Load distributing for locally distributed systems. *Computer*, v.25, n.12, p.33–44.

SILBERSCHATZ, A.; GALVIN, P.; GAGNE, G. (2001). *Sistemas Operacionais: Conceitos e Aplicações*. Campus.

SILVA, D. P. (2003). Usando Replicações para Escalonar Tarefas Bag-of-Tasks em Grids Computacionais. Dissertação (Mestrado), Universidade Federal de Campina Grande (UFCG), Campina Grande, Brasil.

SIMJAVA (2005). Página do pacote de simulação SimJava. Disponível em http://www.icsa.inf.ed.ac.uk/research/groups/hase/simjava/.

SOTOMAYOR, B. (2003). The Globus Toolkit 3 Programmer's Tutorial. Tutorial disponível em http://gdp.globus.org/gt3-tutorial/.

SOUZA, P. S. L. (2000). *AMIGO: Uma Contribuição para a Convergência na Área de Escalonamento de Processos*. Tese (Doutorado), ICMC-USP, São Carlos, Brasil.

STALLINGS, W. (2000). *Computer Organization and Architecture*. Prentice Hall, 5 edição.

TANENBAUM, A. S. (2001). *Modern Operating Systems*. Prentice Hall, 2 edição.

TANENBAUM, A. S.; MAARTEN, V. S. (2002). *Distributed Systems Principles and Paradigms*. Prentice Hall.

TERAGRID (2005). Página do projeto TeraGrid. Disponível em http://www.teragrid.org.

TUECKE, S.; CZAKOWSKI, K.; FOSTER, I.; FREY, J.; GRAHAM, S.; KESSELMAN, C.; MAQUIRE, T.; SANDHOLM, T.; SNELLING, D.; VANDERBILT, P. (2003). Open Grid Services Infrastructure (OGSI) Version 1.0. Relatório técnico, Global Grid Forum. Disponível em http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-29_2003-04-05.pdf.

UNGER, J.; HAYNOS, M. (2003). A Visual Tour of Open Grid Services Architecture. Relatório técnico, IBM. Disponível em http://www-106.ibm.com/developerworks/grid/library/gr-visual.

UNITED DEVICES (2005). Página com Projetos da United Devices. Disponível em http://www.grid.org.

VADHIYAR, S.; DONGARRA, J. (2002). A metascheduler for the grid.

W3C (2005). Página do consórcio World Wide Web. Disponível em http://www.w3c.org.

WOLSKI, R.; SPRING, N. T.; HAYES, J. (1999). The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computing Systems*, v.15, n.5–6, p.757–768.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo