Um Escalonador para Grades Computacionais Utilizando Modelos Econômicos

Fabiano Costa Teixeira

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Um Escalonador para Grades Computacionais Utilizando Modelos Econômicos

Fabiano Costa Teixeira

18 de Dezembro de 2006.

Banca Examinadora:

- Dra. Maria Beatriz Felgar de Toledo (Orientadora)
  IC/ UNICAMP – Universidade Estadual de Campinas
- Dr. Edmundo Roberto Mauro Madeira
  IC/ UNICAMP – Universidade Estadual de Campinas
- Dr. Hélio Crestana Guardia
  UFSCar – Universidade Federal de São Carlos
- Dr. Célio Cardoso Guimarães (Suplente)
  IC/ UNICAMP – Universidade Estadual de Campinas

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Teixeira, Fabiano Costa

T235e Um escalonador para grades computacionais utilizando modelos econômicos / Fabiano Costa Teixeira -- Campinas, [S.P. :s.n.], 2006.

Orientador : Maria Beatriz Felgar de Toledo

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Computação em grade (Sistemas de computador). 2. Processamento distribuído. 3. Modelos econômicos. I. Toledo, Maria Beatriz Felgar de. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Um Escalonador para Grades Computacionais Utilizando Modelos Econômicos

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Fabiano Costa Teixeira e aprovada pela Banca Examinadora.

Campinas, 18 de Dezembro de 2006.

Dra. Maria Beatriz Felgar de Toledo
IC/UNICAMP – Universidade Estadual de Campinas (Orientadora)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 18 de dezembro de 2006, pela Banca examinadora composta pelos Professores Doutores:

Hélio Crestana Guardia

Prof. Dr. Hélio Crestana Guardia
UFSCar.

Edmundo R M Madeira

Prof. Dr. Edmundo Roberto Mauro Madeira
IC – UNICAMP.

Maria Beatriz Felgar de Toledo

Profa. Dra. Maria Beatriz Felgar de Toledo
IC – UNICAMP.

Resumo

A necessidade cada vez maior de um grande poder de processamento e o aumento da complexidade das aplicações motivaram o surgimento de um paradigma de computação distribuído chamado Grade Computacional. Esse paradigma possibilita o compartilhamento de recursos entre participantes heterogêneos, geograficamente separados e sob administrações independentes. No entanto, o comportamento egoísta dos participantes que procuram somente consumir recursos e não fornecer pode dificultar o crescimento e fortalecimento de uma grade computacional.

Diante de tal contexto, essa dissertação propõe a utilização de modelos econômicos no escalonamento de recursos de uma grade computacional. Através da utilização de créditos, os participantes devem, de forma virtual, pagar pela utilização dos recursos remotos.

De maneira a oferecer uma maior flexibilidade, a arquitetura proposta permite ainda a formação de grupos de trabalho em que diversos participantes podem trabalhar de forma conjunta para adquirirem um maior valor de créditos que um participante sozinho não poderia obter.

A arquitetura se preocupa ainda em fornecer garantia de qualidade de serviço, de maneira que um prazo máximo de execução pré-estabelecido por um consumidor seja respeitado. Para essa arquitetura foram desenvolvidos componentes relacionados com a implementação de modelos econômicos e utilizados alguns componentes já existentes no Globus Toolkit.

Abstract

The need for a greater processing power and the increased complexity of applications have caused the appearance of a distributed computing paradigm called Computational Grid. This paradigm allows resource sharing among heterogeneous, geographically distributed and autonomous participants. However, the selfish behavior of participants, that only want to consume resources and do not provide them, may hamper the grid growth.

Thus this dissertation proposes the use of economic models for scheduling resources of the grid. Participants can pay for grid resources using credits.

To offer a greater flexibility, the proposed architecture allows the organization of participants into groups that can work cooperatively to acquire more credits and, thus, to solve problems requiring more credits that a participant alone could not have.

This architecture focuses on providing quality of service. Deadlines established by consumers are guaranteed. The architecture includes components related with the implementation of economic models and others from Globus Toolkit.

Agradecimentos

Agradeço em primeiro lugar a Deus pela vida a mim concedida, por me dar toda a força que sempre precisei para que eu pudesse seguir meus estudos e por permitir que hoje eu esteja concluindo um curso de Mestrado na Unicamp.

Agradeço a meus pais, Vadir e Divina, por todo ensinamento familiar, ético e religioso que me deram. Agradeço por, em inúmeras vezes, se privarem de muitas coisas para me ajudarem nessa grande jornada.

Agradeço a minha orientadora Maria Beatriz Felgar de Toledo por ter me aceito no programa, ter me ajudado tanto nessa fase dos meus estudos com muita dedicação, compreensão e paciência.

Agradeço a minha namorada Thaís por toda compreensão e apoio que me deu durante esse tempo.

Agradeço a minha irmã Eliana, meu cunhado Benedito e minha sobrinha Renata por me acolherem em sua casa, me dando um lar durante minha permanência em Campinas.

Agradeço minhas tias pelas inúmeras orações a mim dedicadas para que eu conseguisse concluir meu mestrado.

Por fim, agradeço a todos os meus colegas de trabalho da PUC Minas – *Campus* Poços de Caldas por todo apoio e incentivo que a mim foram concedidos.

Muito obrigado a todos.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Nesse capítulo será apresentada uma breve introdução do projeto descrito nessa dissertação de mestrado.

1.1 Motivação

O termo Grade Computacional apareceu em meados da década de 90 para denotar um paradigma de computação distribuída voltado ao compartilhamento de recursos [10,11]. Esse compartilhamento de recursos pode ser realizado em um ambiente heterogêneo, geograficamente distribuído e sob administrações independentes. Para empresas com diversas filiais espalhadas pelo mundo, por exemplo, pode ser mais interessante compartilhar recursos ociosos do que adquirir um supercomputador de maior custo.

Embora o paradigma de grade computacional ofereça amplas vantagens com relação ao compartilhamento de recursos, o comportamento egoísta dos participantes (os quais desejam apenas consumir recursos, mas não se preocupam em oferecer os seus) pode prejudicar o crescimento de uma grade. Uma solução para esse problema seria condicionar o uso de um recurso ao oferecimento de outros, da maneira semelhante àquela encontrada na sociedade na qual vivemos, onde para adquirirmos um bem ou serviço temos que oferecer algo em troca. Nesse sentido, torna-se interessante a utilização de Modelos Econômicos dentro do contexto de grades computacionais.

1.2 Objetivos

Esse trabalho visa integrar as áreas de grades computacionais e economia a fim de incentivar a oferta de recursos por parte dos participantes da grade e garantir qualidade de serviço.

Serão apresentadas a proposta e implementação de uma arquitetura baseada em modelos econômicos para o escalonamento e administração de uma grade computacional, de maneira a atender os requisitos citados anteriormente.

O trabalho tem um foco no mecanismo para ajuste automático de preços (considerando fatores econômicos encontrados na Grade Computacional como, por exemplo, oferta, demanda e taxa de utilização de um recurso), na negociação entre um provedor e um consumidor de recursos de forma que ambos sejam capazes de procurar a maximização de seus objetivos e na criação de grupos de trabalho que possibilitem que as organizações disponibilizem diversos recursos para gerar um montante único de créditos a ser utilizado em requisições bastante caras.

Essa arquitetura é genérica o bastante de forma que possa ser integrada a *middlewares* já existentes para a construção de grades computacionais. Assim, deseja-se utilizar o conhecimento já existente e aprimorá-lo, oferecendo recursos adicionais.

De maneira a garantir a continuidade do trabalho foram, na medida do possível, adotados recursos que possam oferecer de maneira simplificada a interoperabilidade com outros produtos. Para isso, faz se uso extensivo de tecnologias Java e serviços *Web*.

1.3 Organização da Dissertação

Essa dissertação está estruturada da seguinte maneira:

- O capítulo 2 apresenta alguns conceitos básicos, referentes a grades computacionais e economia, que serão utilizados durante o decorrer da dissertação.
- O capítulo 3 apresenta alguns trabalhos relacionados que são relevantes à area em questão.

- O capítulo 4 apresenta as plataformas de apoio que serviram como base teórica e/ou prática para o desenvolvimento do projeto.
- O capítulo 5 apresenta a arquitetura para grades computacionais, utilizando modelos econômicos.
- O capítulo 6 apresenta detalhes da implementação da arquitetura proposta.
- O capítulo 7 finaliza a dissertação apresentando as conclusões e contribuíções desse trabalho.

Capítulo 2

Fundamentos

Este capítulo introduz os conceitos básicos relacionados a grades computacionais e a modelos econômicos no gerenciamento de recursos computacionais compartilhados.

2.1 Grades computacionais

Apesar da evolução da tecnologia, a maior complexidade dos problemas a serem resolvidos exige um grande poder de processamento. Um computador pessoal no ano 2001 era tão rápido quanto um supercomputador de 1990, mas a complexidade dos trabalhos realizados também se tornou efetivamente maior, por exemplo, evoluiu da análise de uma estrutura molecular para a análise de uma construção complexa de macromoléculas.

A popularidade da Internet, o avanço e a proliferação das redes de alta velocidade tornaram possível a interligação de diversos recursos computacionais de maneira a compor um único sistema [16]. Na década de 90 foi criado o termo "Grade Computacional" para denotar essa infra-estrutura de sistema distribuído em que recursos (podendo variar desde UCP's e discos até licenças de *software*) geograficamente distribuídos e sob administrações independentes podem ser compartilhados entre os participantes da Grade Computacional.

O agrupamento de recursos distribuídos em uma Grade Computacional permite o compartilhamento de recursos entre os participantes e torna o sistema poderoso o bastante para contribuir para a solução, de forma colaborativa, de diversos problemas sem a

necessidade de os participantes adquirirem novos equipamentos. Ao redor de todo o mundo é possível encontrar um grande número de computadores cuja capacidade não é totalmente explorada. Dessa forma, o agrupamento dos percentuais de ociosidade desses equipamentos pode oferecer um poder computacional considerável para a solução de muitos problemas.

Como um exemplo desse paradigma, o projeto *SETI@home* [13] faz uso de uma Grade Computacional que utiliza os ciclos ociosos de computadores espalhados por todo o mundo para analisar dados referentes a sinais de rádio recebidos por um telescópio localizado em Porto Rico em busca de vida inteligente fora do planeta Terra. Os recursos dos participantes dessa Grade Computacional formam uma estrutura computacional com uma capacidade coletiva de processamento de dezenas de *Teraflops* [14].

2.1.1 Organizações virtuais

Um dos principais problemas de uma Grade Computacional é o gerenciamento de recursos [11]. Universidades, empresas e outras instituições independentes e sob administrações distintas necessitam ditar as regras sob as quais seus recursos podem ser compartilhados. Esse conjunto de instituições que compartilham recursos sob tais regras forma o que é chamado de Organização Virtual.

As Organizações Virtuais podem ser compostas para atender diversos tipos de trabalhos e podem variar em muitos aspectos como: tamanho, finalidade à qual se destinam, escopo, duração, entre outros [11]. Os participantes de uma Organização Virtual definem regras como: quem pode usar recursos, quais recursos podem usar, quando e como.

2.1.2 Arquitetura de grades

A arquitetura de uma Grade Computacional fornece mecanismos para o compartilhamento de recursos [11] atendendo aspectos como interoperabilidade e segurança.

Durante uma década de pesquisa nessa área, foi produzido um considerável consenso quanto aos requisitos e à arquitetura de uma Grade Computacional [15]. Muitos protocolos para troca de mensagens entre aplicações foram propostos, assim como diversas API's com o intuito de oferecer meios mais simples e produtivos para a construção de uma Grade Computacional.

Esses protocolos e API's podem ser agrupados em uma estrutura de camadas de acordo com o papel que exercem [15]. As camadas internas possuem um número menor de elementos e, por conseqüência, a representação da arquitetura através do modelo de camadas possui uma forma parecida com uma ampulheta (figura 2.1).

A camada Ambiente acomoda os diversos recursos da grade que são compartilhados, através da utilização de protocolos. Tais recursos podem ser tanto recursos físicos (unidades de disco, processadores, entre outros) como também recursos lógicos (como sistemas de arquivo e licenças de software).

Os componentes da camada Ambiente oferecem protocolos e mecanismos que implementam as operações fornecidas pelos recursos. Esses protocolos e mecanismos podem ser fornecidos pelo fabricante do recurso ou até mesmo disponibilizados pelo *Middleware* da Grade Computacional.

A camada Conectividade e Segurança define um conjunto básico de protocolos para atender aos requisitos de comunicação e autenticação de uma operação da Grade Computacional. Essa camada é composta por protocolos como, por exemplo, IP, ICMP, TCP, UDP, entre outros.

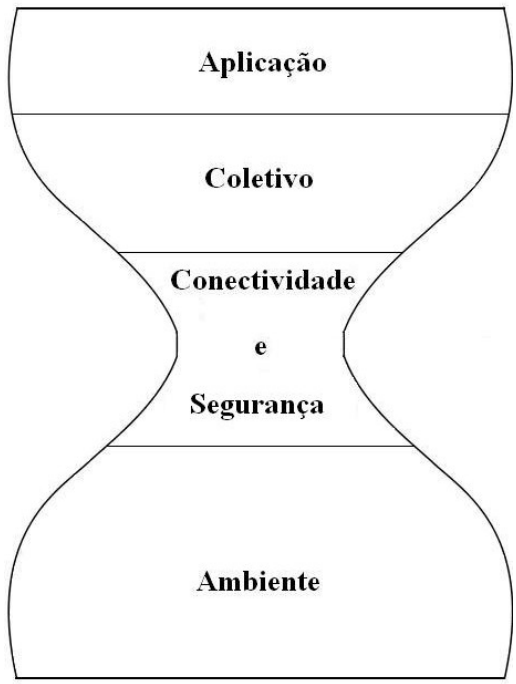


Figura 2.1: Modelo de Camadas de uma Grade Computacional

Os aspectos de segurança são implementados por mecanismos de autenticação e criptografia. Em uma Organização Virtual os mecanismos de autenticação devem oferecer, entre outras, as seguintes características:

- **Único *Log-on*:** Após o usuário efetuar um processo de autenticação para utilizar um determinado recurso da Grade Computacional, deve ser possível que ele utilize diversos outros recursos sem a necessidade de nova autenticação;
- **Delegação:** É necessário que um usuário possa atribuir a um determinado programa os seus direitos de maneira que esse atue no sistema em seu nome. Esse programa também deve ser capaz de atribuir esses direitos a um outro;
- **Integração com várias soluções locais de segurança:** Vários sistemas podem utilizar sistemas locais de segurança e é necessário que os mecanismos da Grade sejam capazes de interagir com eles.

A camada Conectividade e Segurança utiliza os mecanismos da camada Ambiente para realizar de forma segura as operações sobre um recurso compartilhado (como por exemplo: negociação, monitoração, contabilização, entre outros).

Os protocolos existentes na camada Conectividade e Segurança incluem:

- **Informação:** Os protocolos encontrados nessa classe são utilizados para obter informações sobre um determinado recurso;
- **Gerenciamento:** Responsáveis pela negociação de utilização de um determinado recurso incluindo aspectos como qualidade de serviço, operação a ser executada, entre outros.

A camada Coletivo possui protocolos e API's que implementam um conjunto de operações sobre uma coleção de recursos. É possível destacar alguns exemplos:

- **Serviços de diretório:** Armazenam e disponibilizam informações sobre os recursos existentes de forma que os participantes possam consultá-los com a finalidade de localizar e obter dados sobre esses recursos;
- **Serviços de alocação e escalonamento:** Alocam um recurso apropriado para atender uma determinada requisição em função de certos parâmetros.

Na camada Aplicação são encontradas as aplicações propriamente ditas que operam utilizando recursos existentes em uma Grade Computacional.

2.2 Economia

O estudo da economia permite analisar e prever a forma pela qual os participantes de uma sociedade interagem entre si naquilo que diz respeito à troca de serviços ou bens.

Na ciência econômica encontramos duas teorias [3]: a teoria macroeconômica e a teoria microeconômica. A primeira tem uma preocupação com o sistema econômico como um todo enquanto a segunda trata das atividades econômicas individuais envolvendo consumidores, proprietários de recursos, entre outros. É possível representar um sistema econômico de forma circular simplificada mostrando o fluxo de dinheiro, de bens e de

serviços. Nesse fluxo circular são encontradas as unidades: famílias e empresas, as quais interagem entre si comercializando produtos, serviços ou bens e mão de obra. A figura 2.2 ilustra esse fluxo.

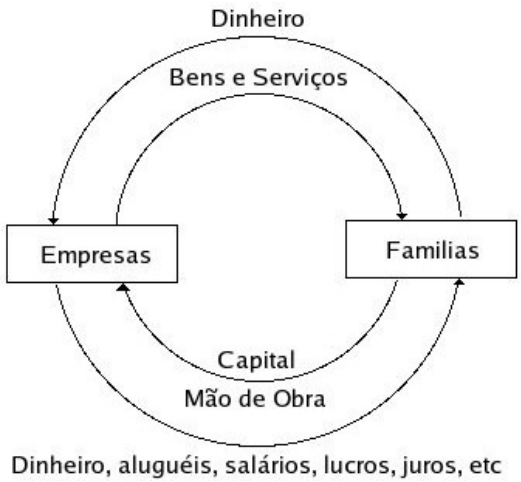


Figura 2.2: Fluxo circular de uma economia.

Um sistema econômico precisa desempenhar cinco funções [3]:

- Determinar o que se deve produzir;
- Determinar como se deve organizar a produção;
- Determinar como se deve distribuir a produção;
- Determinar como se deve racionar bens;
- Determinar como sustentar e expandir a capacidade produtiva

2.2.1 Recursos

Em uma economia são encontrados diversos tipos de recursos [2] necessários para a produção de bens ou serviços que representam aquilo que os participantes dessa economia dão valor (como exemplo, o ser humano é um recurso necessário para o oferecimento de um serviço). Os recursos possuem três características básicas [3]: a maioria dos recursos

são limitados em quantidade, são versáteis e podem ser combinados com a finalidade de produzir determinados bens ou serviços. Tais recursos podem ser escassos (ou seja, com quantidade limitada) e são classificados como *Recursos Econômicos* ou podem ser tão abundantes (que não possuam preços) e classificados como *Recursos Livres.*

O conjunto dos recursos disponíveis em uma economia é chamado de Dotação de Recursos [2] e a forma pela qual esses recursos podem ser empregados para a produção de bens ou serviços é chamada de Tecnologia. No entanto, um mesmo recurso pode ser empregado de formas diferentes para produzir bens ou serviços diferentes, mas essas diversas formas são descritas pela Tecnologia.

2.2.2 Preferências dos Indivíduos

Na sociedade é possível verificar que, de alguma forma, os indivíduos que a compõe ditam quais bens devem ou não ser produzidos, qual a quantidade e a periodicidade deles. Isso se dá em função das *Preferências dos Indivíduos* e essas podem ser ordenadas de acordo com um grau. Assim, cada indivíduo possui uma classificação de preferências partindo daquela de maior para a de menor grau e ele buscará atender a preferência de maior grau possível [2].

2.2.3 Preços

Em um sistema econômico, serviços ou bens são trocados ou comercializados entre os participantes dessa economia. Dessa forma, esses bens ou serviços precisam de um preço de venda ou uma proporção a ser utilizada em casos de troca. Esse preço varia automaticamente de acordo com a concorrência existente no mercado [4]: se a quantidade de bens ou serviços oferecidos pelos vendedores for maior que a quantidade exigida pelos compradores, os vendedores concorrem entre si de forma a oferecer um menor preço e melhores condições para atrair os compradores. Se a quantidade requerida for maior que a

ofertada, os compradores se propõem a pagar um preço mais alto para que possam adquirir tais produtos ou serviços.

Caso essa concorrência funcione melhor ou pior, essa variação do preço dos bens pode ocorrer de uma forma mais ou menos rigorosa. Uma forma de mercado em que o sistema de concorrência é mais organizado é aquela em que todas as compras e vendas são anunciadas (como, por exemplo, em um pregão) e todos os compradores têm a possibilidade de aumentar os lances e os vendedores diminuir seus preços.

A oferta e a demanda (assunto da próxima subseção) influenciam os preços dos bens ou serviços. A fórmula para geração de preços considera intervalos de tempo que servem como base para o cálculo [8]. Nessa fórmula, são conhecidos o preço p do bem ou serviço i para um intervalo n (denotado por p^i_n), a demanda no intervalo (denotada por d^i_n) e a oferta (denotada por S^i). Assim, é possível que no término de um intervalo de tempo, o preço do bem ou serviço seja recalculado através da fórmula:

$$p^i_{n+1} = p^i_n + c\left(\frac{(d^i_n - \alpha.S^i)}{(\alpha.S^i)}\right)$$

Com essa fórmula o preço para um determinado período é igual ao preço do período anterior adicionado de uma correção que pode ser positiva ou negativa. A oferta disponível é igual à oferta total multiplicada pela constante a que varia entre 0 e 1. Dessa forma, quando a demanda for igual à oferta disponível, é encontrado o preço de equilíbrio. A constante positiva c determina a velocidade na qual o preço deve ser ajustado[8].

2.2.4 Oferta e Demanda

A oferta é a quantidade de um bem ou serviço que os produtores ou vendedores desejam vender. Algumas variáveis influenciam a oferta de determinado bem ou serviço, sendo assim a função geral da oferta é [9]:

$q_i = f(p_i, p, p_n, O)$

em que:

q_i = quantidade ofertada do bem i;

p_i = preço do bem i;

p = preço dos fatores e insumos de produção;

p_n = preços de outros n bens que concorrem com o bem em questão;

O = objetivos e metas do produtor ou vendedor.

A demanda é a quantidade de um bem ou serviço que os compradores ou consumidores desejam adquirir. Sendo assim, a demanda pode ser representada pela função [9]:

$q_i = f(p_i, p_s, p_c, R, G)$

em que:

q_i = quantidade procurada;

p_i = preço do bem;

p_s = preço dos bens que concorrem com o bem em questão;

p_c = preço dos bens complementares;

R = renda do consumidor;

G = gostos, hábitos e preferências do consumidor.

O comportamento da demanda e da oferta em relação uma à outra influencia o comportamento da variação do preço [4]. Se a demanda e a oferta de um determinado bem ou serviço forem iguais é atingido o que é chamado de Equilibrio de Mercado em que o preço é estacionário. Se a demanda é maior que a oferta, os compradores tendem a aumentar o valor do lance ocasionando assim o aumento do preço. Se a oferta for maior que a demanda, os vendedores tendem a diminuir os preços cobrados ou colocar o bem ou serviço em liquidação de forma a atrair compradores, tendo assim uma queda do preço.

O gráfico ilustrado na figura 2.3 demonstra o comportamento da oferta, demanda e dos preços.

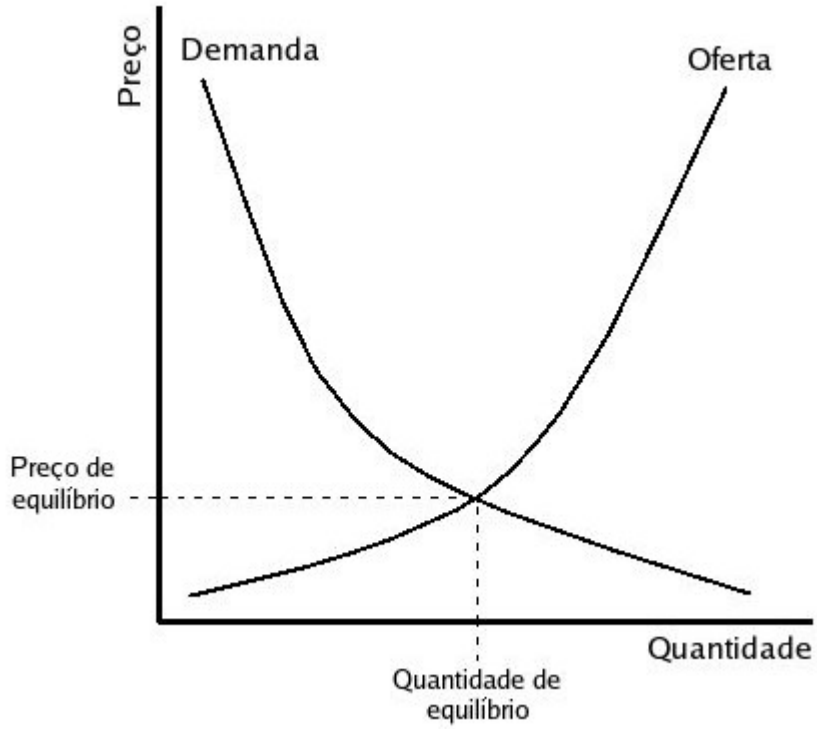


Figura 2.3: Gráfico da oferta e da demanda.

2.2.5 Critério de Pareto

Em uma economia existem diversos participantes, estejam eles efetuando o papel de vendedores ou compradores, e durante todo o tempo eles estão procurando otimizar seus objetivos de forma a conseguir um melhor proveito individual. As escolhas feitas por um indivíduo formam, coletivamente, o que é chamado de estado social [2] (ou seja, de acordo com as escolhas e o comportamento dos indivíduos são alcançadas configurações diferentes da economia). É bastante importante e útil a existência de uma forma de comparar e colocar em escala esses estados e uma das formas mais utilizadas pelos economistas é o Critério de Pareto: *ao comparar dois estados sociais, I e J, o estado I é Pareto-Preferível ao estado J se ninguém no estado I estiver pior que no estado J e se pelo menos uma pessoa estiver melhor no estado I do que no estado J*.

O melhor de todos os estados possíveis é chamado de Pareto-Ótimo. Esse critério pode ser definido da seguinte maneira: *um estado social é Pareto-Ótimo se nenhum outro estado social obtenível for Pareto-Preferível a ele*.

2.2.6 Estruturas de Mercado

Diferentes estruturas de mercado podem influenciar a determinação de preços e da produção e normalmente elas são condicionadas por três variáveis principais [9]: número de firmas produtoras, diferenciação do produto e existência de barreiras a entradas de novas empresas.

Na estrutura de Concorrência Perfeita são identificadas as seguintes hipóteses [9]:

- **Atomicidade:** o mercado é formado por inúmeros vendedores e compradores, de forma que a entrada ou saída de um vendedor ou comprador não influencia a economia como um todo.
- **Homogeneidade:** todos os bens ou serviços produzidos pelas empresas são homogêneos, ou seja, não possuem diferenças que levem o comprador a ter preferência por um ou outro produto.
- **Mobilidade de agentes:** O mercado não apresenta restrições quanto à entrada ou saída de vendedores ou compradores.
- **Racionalidade:** Os vendedores sempre buscam maximizar seus lucros e os compradores sua satisfação.
- **Transparência de mercado:** Todos os participantes do mercado, sejam eles compradores ou vendedores, possuem livre acesso a informações sobre preços, custos, lucros dos seus concorrentes.
- **Mobilidade de bens:** Não é considerada a localização geográfica dos vendedores e dos consumidores, ou seja, a distância não influencia os preços. Dessa forma, o comprador não possui preferência em função da localidade do vendedor.

A estrutura de mercado de Concorrência Imperfeita é mais próxima da realidade devido a algumas características como:

- Existem muitas empresas produzindo um determinado bem ou serviço;
- Cada empresa possui liberdade para produzir um bem ou serviço diferenciado de forma a atrair clientes;
- Cada empresa tem poder para decidir sobre os preços de seus bens ou serviços, pois é possível que sejam diferenciados em relação à concorrência.

Essa diferenciação do bem produzido ou do serviço prestado varia de acordo com o vendedor ou fornecedor, podendo ser, por exemplo, a qualidade do bem oferecido, alguma característica técnica específica como também algum artifício promocional oferecendo brindes aos compradores, entre outros.

O Oligopólio é uma estrutura em que o domínio do mercado está condicionado a poucos vendedores, podendo ser de duas formas:

- **Concentrado:** Estrutura encontrada em um setor em que exista um pequeno número de vendedores. Dessa forma o comprador não tem outra escolha de compra que não seja um deles.
- **Competitivo:** Nessa estrutura existe um grande número de empresas, as quais podem ser escolhidas pelo comprador. No entanto, o domínio do mercado (pela preferência do consumidor) está sob um número bastante pequeno de vendedores.

A estrutura de Monopólio é encontrada em situações em que apenas uma empresa domina o mercado sem que os compradores tenham opção de escolher outros vendedores. Esse tipo de estrutura proporciona ao vendedor um controle bastante grande sobre o preço dos bens ou serviços produzidos, oferecendo assim mecanismos para que esse maximize seus lucros.

2.3 Grades Computacionais e Modelos Econômicos

As grades computacionais foram concebidas com o intuito de proverem o compartilhamento de recursos heterogêneos que estejam remotamente distribuídos e sob administrações independentes. Sendo organizados dessa forma, os integrantes de uma grade podem agir sob os papéis de consumidores e provedores [1, 16, 18]: aqueles que utilizam recursos de outros participantes e aqueles que os disponibilizam, respectivamente.

É possível efetuar uma analogia entre esse paradigma computacional e a sociedade real atual, onde existem fornecedores de mercadorias ou serviços e consumidores, sendo uns clientes dos outros. Se esse modelo de sociedade não fosse de alguma forma controlado, permitindo que qualquer pessoa pudesse consumir qualquer produto ou serviço sem a necessidade de oferecer nada em troca, poderia ocorrer um comportamento egoísta por parte dos participantes da sociedade, tendo cada um o interesse maior em consumir do que em produzir ou ofertar. Esse tipo de comportamento poderia levar a um desinteresse em produzir, tornando a sociedade enfraquecida ou até mesmo impraticável.

Essa mesma característica pode ser encontrada em uma grade computacional. A existência de participantes com interesse em apenas consumir recursos da grade pode prejudicar o crescimento e a evolução dessa devido a uma possível baixa oferta de recursos. Uma solução para esse problema é um modelo semelhante ao empregado na sociedade real em que o consumo de um recurso estaria de alguma forma condicionado à oferta de um outro. Esse modelo incentiva os participantes a compartilhar seus recursos ociosos de forma a gerar créditos para poder utilizar os recursos dos demais participantes tornando a grade maior e mais poderosa.

Com a utilização de um modelo econômico em uma grade computacional, o compartilhamento de recursos fica semelhante ao comércio de um produto ou serviço qualquer efetuado na sociedade real: os recursos compartilhados devem possuir preços e os consumidores devem pagar pelo uso do recurso. Esses preços podem ser uma representação virtual de créditos que são válidos no ambiente da grade computacional tendo em vista o controle dos recursos como também poderia ser um valor monetário real.

2.3.1 Qualidade de Serviço

Uma grade computacional tem uma grande variedade de recursos compartilhados e usuários interessados em utilizar tais recursos [16]. A administração desse ambiente é uma tarefa complexa que exige mecanismos capazes de analisá-lo e tomar as decisões de acordo com a política adotada, podendo ser:

- **Política centrada no sistema:** Nessa abordagem, o gerenciamento do sistema se preocupa com a otimização do sistema como um todo baseando-se em aspectos como *throughput*.
- **Política centrada no usuário:** Nessa política existe uma preocupação em atribuir ao usuário recursos que sejam capazes de satisfazer seus requisitos de processamento, como, por exemplo, qualidade de serviço (*QoS).*

Um sistema com política centrada no usuário deve ser capaz de promover recompensas e penalidades aos usuários de maneira a controlar seu comportamento [16]. Na requisição de um recurso para a execução de uma tarefa, o usuário pode estabelecer critérios como, por exemplo, o prazo de execução [18]. No entanto, o tempo de processamento de uma determinada tarefa é relacionado ao poder de processamento despendido a ela, ou seja, quanto menor for o prazo de execução maior o poder computacional necessário. Se o valor do prazo de execução não for de certa maneira controlado, pode haver uma tendência de os usuários forçarem prazos sempre pequenos (muitas vezes, menores que o necessário) o que poderia causar uma sobrecarga do sistema.

É comum encontrar abordagens centradas no usuário que utilizam modelos econômicos no controle dos recursos [16].

2.3.2 Determinação de Preços

Em uma grade computacional controlada por modelos econômicos, os provedores de recursos devem possuir mecanismos para determinar valores (preços) a serem cobrados pela utilização desses recursos [16].

Como em uma economia real, muitos parâmetros podem ser considerados no momento da geração dos preços para os recursos. Em sistemas em que parâmetros como oferta, demanda e QoS não são considerados o mecanismo de geração de preços é bem mais simples [18]. No entanto, sistemas mais complexos e precisos podem ser desenvolvidos utilizando diversos parâmetros:

- **Oferta e demanda:** Através dessas informações é possível estabelecer preços que sejam condizentes com a situação atual do mercado do recurso. Por exemplo, se a demanda por um determinado recurso cai, seu preço tende a diminuir. Se a demanda cresce, o preço tende a aumentar;
- **Duração da utilização:** O tempo durante o qual o recurso permanecerá alocado para a requisição;
- **Momento da utilização:** Através do monitoramento de um recurso é possível identificar os momentos de pico e de baixa utilização desse. Dessa forma, os preços de um recurso podem variar em função do momento de sua utilização. Os preços podem abaixar na madrugada e finais de semana, por exemplo;
- **Lealdade dos consumidores:** Consumidores que possuem um volume de utilização grande podem obter preços diferenciados.

2.3.3 Localização e Negociação por Recursos

No momento em que um determinado cliente necessita utilizar um recurso da grade, ele deve descrever o tipo de recurso a ser utilizado, as características que esse recurso deve ter, o prazo para execução, o preço máximo a ser pago, entre outros [16]. Com base nessas informações é iniciado um processo para localizar (por exemplo, em um serviço de diretório) os recursos que atendam a tais especificações e selecionar qual recurso deve ser utilizado. Essa análise é efetuada por um componente denominado *mediador* [1] e envolve uma negociação com os provedores de recursos.

[1] Do inglês *broker*

Durante essa negociação entre o mediador e o provedor do recurso, ambos os lados podem efetuar alterações nos parâmetros (aumento do preço a ser pago, relaxamento do prazo de execução, entre outros) até que seja encontrado um ponto interessante para ambos ou até que seja atingido um ponto a partir do qual a negociação não obterá sucesso.

O mediador pode utilizar critérios de otimização dependendo da necessidade do usuário. Entre outros, esses critérios podem ser:

- **Otimização de prazo:** O *mediador* procura um recurso que ofereça o menor prazo de execução possível sem que o valor cobrado ultrapasse o estipulado pelo cliente;
- **Otimização de valor:** O *mediador* procura um recurso que ofereça o menor valor possível sem que o prazo de execução seja ultrapassado.

Para que a exposição de recursos com seus respectivos preços possa ser estabelecida, deve ser adotado um modelo econômico [12]. Alguns modelos são discutidos a seguir.

No Modelo de Mercado de Produto, o vendedor define o preço de um bem ou serviço e cobra dos compradores ou consumidores pela utilização desse. Esse preço pode ser fixo ou variável de acordo com a demanda. Nesse segundo caso, o preço aumenta com o aumento da demanda e diminui com a queda dessa.

O Modelo de Preço Afixado é bastante parecido com o anterior, no entanto os vendedores ou fornecedores lançam promoções de forma a atrair novos compradores. Essas promoções visam aumentar o consumo do bem ou serviço em determinadas circunstâncias, como, por exemplo, em momentos em que a procura é menor (por exemplo: feriado, final de semana, madrugada).

Nos modelos anteriores, os vendedores definem os preços que serão cobrados pelo uso de um determinado bem ou serviço. No Modelo de Negociação, os compradores precisam negociar diretamente com os fornecedores de forma a encontrar um preço que seja do interesse de ambas as partes ou até que seja detectado que esse preço não será atingido e a negociação deve ser abortada. Os negociantes possuem suas funções objetivo que ditam qual é a meta da negociação. O vendedor procura vender ou alocar a maior quantidade possível de um bem ou serviço pelo maior preço possível, no entanto o comprador procura

adquirir os bens ou serviços que atendam suas necessidades da melhor forma com o menor preço.

O Modelo de Oferta/Contrato é bastante conhecido e utilizado pelas empresas em processos de aquisição de bens ou serviços. Quando um consumidor deseja adquirir algo, ele anuncia essa intenção e aguarda pelo recebimento de ofertas dos vendedores ou fornecedores. Ao receber essas ofertas, o consumidor as analisa e verifica qual delas satisfaz seu interesse da melhor forma.

O Modelo de Leilão envolve uma negociação de um vendedor para n compradores. É bastante conhecido e utilizado na sociedade atual na venda de bens e possui algumas formas de ser conduzido:

- **Leilão Inglês:** O vendedor começa o processo com um valor inicial para o bem ou serviço a ser vendido e todos os compradores são livres para aumentarem suas ofertas em relação aos outros de forma a oferecer o melhor valor ao vendedor. Esse processo termina quando nenhum comprador lança mais nenhuma oferta.
- **Primeiro preço:** Todos os compradores enviam uma oferta sem ter conhecimento das ofertas dos demais compradores. Ao receber todas as ofertas, o vendedor verifica qual é a maior e efetua a venda para quem fez a oferta.
- **Segundo preço:** Todos os compradores enviam uma oferta sem ter conhecimento das ofertas dos demais compradores. Ao receber todas as ofertas, o vendedor verifica qual é a maior e efetua a venda para quem a fez, porém com o preço da segunda maior oferta.
- **Leilão Holandês:** O vendedor inicia o leilão com um preço bastante alto e vai diminuindo-o até que algum comprador se disponha a adquirir o bem ou serviço pelo preço atual.

No Modelo de Cooperativa, é formada uma cooperativa na qual os integrantes compartilham bens ou serviços entre si. Dessa forma, quem dispõe de algo para ser compartilhado pode ter acesso aos demais bens ou serviços existentes na cooperativa. No

entanto, é necessário um modelo bastante sofisticado de forma a controlar o quanto cada participante pode utilizar.

Os modelos apresentados anteriormente são referentes a situações onde existe a concorrência pela venda de produtos ou serviços, ou seja, um mercado competitivo. No Monopólio, apenas um vendedor possui um determinado bem ou serviço a ser comercializado, sendo assim ele define por si próprio qual o preço do bem ou serviço.

2.3.4 Contabilização e Pagamento

Em uma grade computacional em que modelos econômicos são utilizados, é de fundamental importância a existência de um mecanismo de pagamento. Através desse mecanismo, são efetuados o controle do saldo de cada um dos participantes e a transferência de créditos referentes à utilização de um recurso compartilhado.

Quando um determinado participante deseja utilizar um recurso, ele deve possuir créditos suficientes para pagar por esse recurso. As formas de pagamento podem ser [20]:

- **Pré-pagas:** o usuário efetua o pagamento pelo acesso a um recurso antes de iniciar a utilização desse;
- **Pós-pagas:** o usuário efetua o pagamento após a utilização do recurso.

Para que um mecanismo de controle de créditos e pagamentos seja aplicado a uma grade computacional, alguns requisitos devem ser observados [19]:

- **Segurança:** As transações de transferência de créditos entre os participantes devem trafegar pela Internet de forma segura.
- **Confiabilidade:** O mecanismo de pagamento e controle de créditos deve estar sempre operante para permitir o compartilhamento de recursos.
- **Escalabilidade***:* O ingresso de novos participantes não deve degradar o desempenho do sistema.

- **Aceitação:** Todos os participantes da grade devem aceitar o mesmo mecanismo de pagamento. Dessa forma, os créditos podem ser utilizados para o pagamento de qualquer recurso.
- **Base de consumidores:** Refere-se ao número de participantes que, potencialmente, podem utilizar e pagar pelo consumo de recursos. Esse número influi na aceitação de um mecanismo de pagamento por parte de um determinado provedor de recursos: quando existe um número considerável de participantes adotando um determinado mecanismo de pagamento, novos provedores se esforçarão para adotar o mesmo mecanismo.
- **Eficiência:** A utilização freqüente de recursos compartilhados pode gerar uma quantidade considerável de pagamentos de pequenos valores a serem efetuados. O mecanismo para geração de pagamentos deve efetuá-los sem que haja uma degradação de desempenho.
- **Facilidade de uso:** O mecanismo de pagamento deve fornecer formas simples para que o usuário efetue seus pagamentos sem que sejam solicitadas ao usuário, constantemente, informações sobre as transações. No entanto, os usuários devem ser capazes de informar o valor máximo que desejam gastar e autorizar pagamentos acima desse valor.

Os mecanismos de pagamento podem ser agrupados nas seguintes classes [19]:

- **Moeda eletrônica:** Um determinado usuário que deseja efetuar um pagamento através desse modelo deve comprar (através de cartões de créditos, depósitos, entre outros) um certificado eletrônico de moeda que representa um determinado valor. No momento do pagamento, esse usuário apresenta esse certificado para que o credor possa depositá-lo em sua própria conta. É necessário que os servidores que coordenam as operações de crédito e débito evitem as tentativas, por parte dos usuários, de utilizar o mesmo certificado eletrônico de moeda mais de uma vez.

- **Crédito e Débito:** Em um sistema que utiliza a abordagem de crédito e débito, cada usuário possui uma conta com saldo positivo em um determinado servidor. As contas de consumidores e provedores são atualizadas conforme as transações efetuadas.
- **Apresentação segura de cartão de crédito:** O usuário que deseja efetuar um pagamento possui um número de cartão de crédito. Esse número é criptografado utilizando uma chave pública e só pode ser lido pelo credor ou, dependendo da abordagem, por um terceiro que irá intermediar tal operação.

Capítulo 3

Trabalhos Relacionados

Este capítulo é dedicado à apresentação de trabalhos na área de grades computacionais.

GRACE

GRACE (*Grid Architecture for Computational Economy*) [20] é uma arquitetura para gerenciamento de recursos em uma Grade Computacional baseada na utilização de modelos econômicos. Ela utiliza *middlewares* já existentes para grades computacionais (como o Globus [25], por exemplo), mas oferece novos serviços.

Essa arquitetura auxilia tanto os provedores quanto os consumidores de recursos a alcançar seus objetivos. Os consumidores definem os requisitos necessários para a execução de um determinado trabalho (como, por exemplo, prazo de execução e valor máximo a pagar) através de agentes responsáveis por localizar os recursos e definir aquele que melhor atenda os interesses dos usuários. Os provedores de serviços fazem uso dos serviços disponibilizados por *GRACE*, através de API's, de maneira a estabelecer os preços de seus recursos, as políticas e regras de utilização. Os módulos da arquitetura (figura 3.1) são: *Grid Consumer*, *Grid Resource Broker*, *Grid Middleware Services* e *Grid Service Providers*.

O módulo *Grid Resource Broker* (GRB) funciona como um intermediário entre os usuários e os recursos compartilhados [20, 21]. Entre outras funções, ele é responsável por

localizar os recursos, selecionar qual atende ao usuário da melhor forma e iniciar sua execução. No GRB são identificados os seguintes componentes:

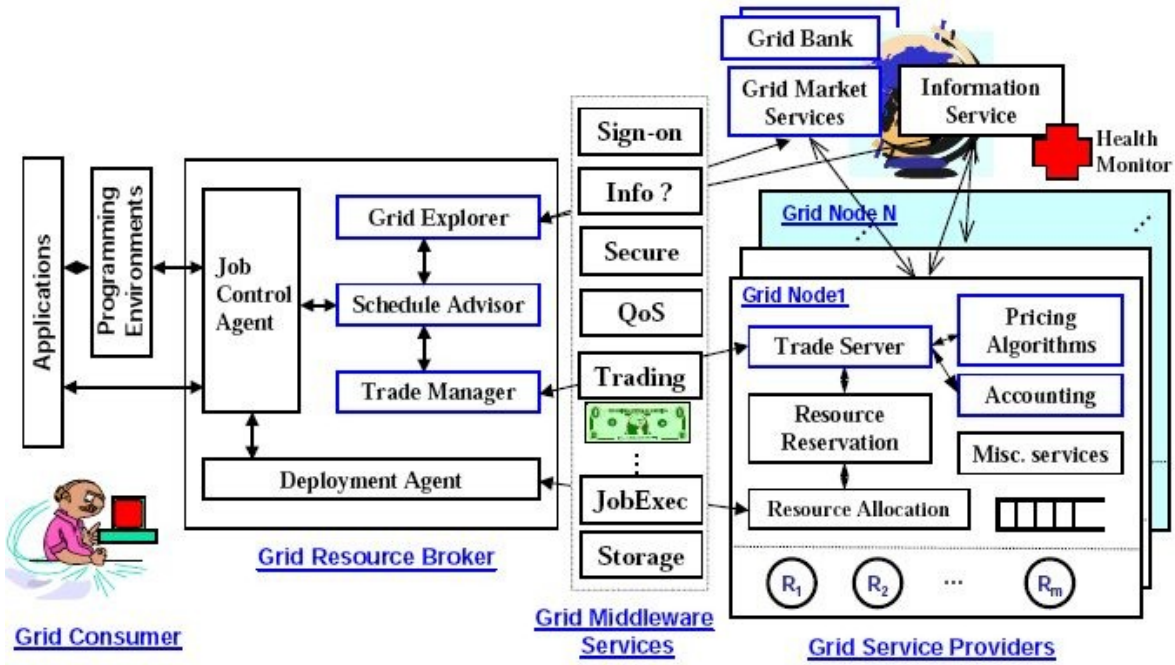


Figura 3.1: Arquitetura *GRACE*

- ***Job Control Agent*:** Efetua o tratamento da tarefa a ser executada. Interage com o *Schedule Advisor* de maneira a coordenar o processo de escalonamento e com o *Deployment Agent* de forma a coordenar a execução das tarefas. Efetua ainda o monitoramento da tarefa e interage com o usuário;

- ***Schedule Advisor*:** Tem o papel de efetuar o escalonamento das tarefas. Através do *Grid Explorer,* ele localiza os recursos existentes na grade e atribui tarefas a recursos obedecendo aos critérios estipulados pelo usuário;

- ***Grid Explorer*:** Responsável pela localização de recursos através da interação com serviços de informações;

- ***Trade Manager*:** Responsável por negociar a utilização de um recurso com o provedor de recursos. Para efetuar tal função ele interage com serviços de negociação oferecidos pela camada de serviços do provedor;

- ***Deployment Agent*:** Responsável por iniciar a execução de uma determinada tarefa de acordo com a decisão tomada pelo escalonador. Possui ainda a função de receber o estado de tal execução informando o *Job Control Agent*.

O *Grid Middleware* contém os serviços para que um usuário execute uma aplicação de forma remota. Inclui funções de segurança, autenticação, transferência de dados, controle de execução, serviços de informações, entre outros. Sendo assim, *GRACE* oferece mecanismos para que usuários e provedores negociem os recursos da grade. Os provedores de serviços interagem com os seguintes componentes:

- ***Trade Server*:** Responsável por maximizar os lucros obtidos pelo provedor, ele negocia com os clientes interessados em utilizar os recursos disponíveis. Dessa maneira ele necessita consultar as políticas de preços estabelecidas de maneira a conduzir tal negociação e solicitar o armazenamento dos dados referentes às utilizações de recursos para que essas possam ser cobradas do usuário.

- ***Princing Polices*:** São as políticas a serem utilizadas para definir os preços que o provedor deve cobrar dos clientes pela utilização de recursos. Esses preços podem variar de acordo com o momento de utilização, cliente que está utilizando, oferta, demanda, entre outros.

- ***Resource Acconting and Charging*:** Armazena informações referentes às utilizações de recursos efetuadas pelos clientes para fins de cobrança.

3.2 Nimrod/G

O *Nimrod/G* [17] é uma ferramenta utilizada para efetuar experimentos paramétricos em uma Grade Computacional. Ele oferece uma linguagem declarativa que possibilita ao usuário modelar o experimento a realizar. Dessa forma, a ferramenta faz uma varredura desses parâmetros e distribui a simulação entre diversas máquinas da grade.

Para efetuar a distribuição do processamento entre os diversos participantes de uma grade computacional, *Nimrod*/G utiliza um sistema de escalonamento econômico [17] [21].

Dessa forma, o usuário especifica o tempo dentro do qual o processamento remoto deve ser concluído e também o valor máximo a ser gasto com tal operação.

O *Nimrod/G* utiliza três algoritmos de escalonamento, cada um com um tipo de estratégia de otimização [21]:

- **Custo:** O algoritmo seleciona o recurso que ofereça o serviço pelo menor custo respeitando o prazo de execução*;*
- **Prazo de execução**: O algoritmo seleciona o recurso que conclua o serviço no menor tempo sem que o limite de custo seja ultrapassado;
- **Conservativo:** O algoritmo procura minimizar o prazo de execução respeitando o valor máximo a ser gasto. No entanto, ele garante um determinado valor a ser gasto para todas as tarefas não processadas.

O *Nimrod*/G, conforme ilustrado na figura 3.2, possui os seguintes componentes:

- ***Nimrod/G Client*:** Oferece a interação do sistema com o usuário. Através dessa interface o usuário pode variar os parâmetros referentes a prazo de execução e preço e monitorar o andamento do experimento;
- ***Parametric Engine*:** Através do modelo do experimento recebido, ele efetua a varredura dos parâmetros e criação das tarefas a serem distribuídas pela grade. Através de interfaces com o *Schedule Advisor* e com o *Dispatcher,* ele obtém informações sobre o recurso a ser utilizado para determinada tarefa e solicita que essa seja executada. Ele é responsável ainda por armazenar o estado da tarefa de forma permanente;
- ***Scheduler*:** Através de consultas em serviços de informações da grade, ele localiza os recursos que potencialmente podem atender os requisitos e, utilizando os algoritmos de escalonamento, determina qual recurso deve ser utilizado para executar determinada tarefa em função dos parâmetros de prazo de execução e limite de preço;

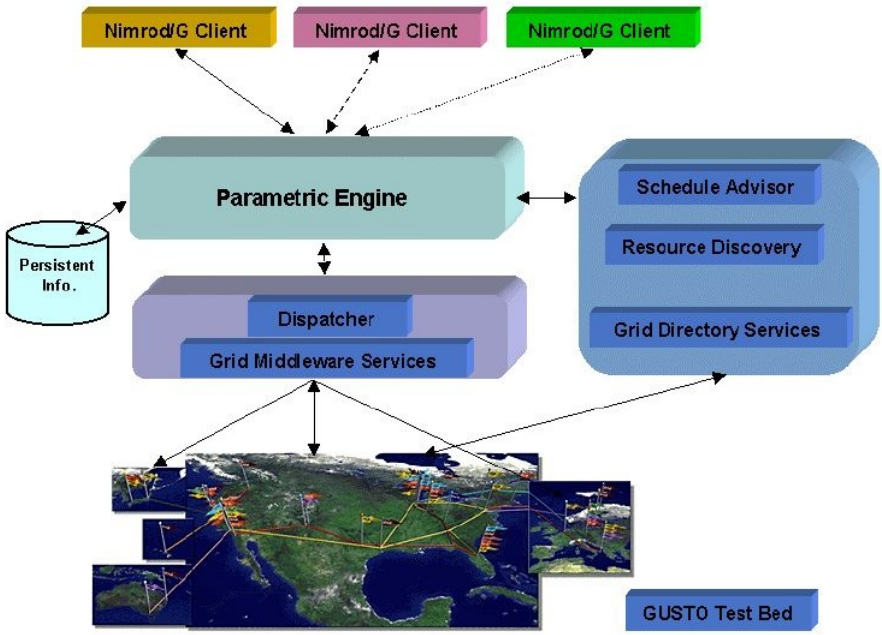


Figura 3.2: Arquitetura do *Nimrod*/G

- ***Dispatcher*:** Através da decisão tomada pelo escalonador, o *Dispatcher* deve iniciar a execução de uma determinada tarefa e informar o estado da execução ao *Parametric Engine*;
- ***Job Wraper*:** Mecanismo que é iniciado no servidor a pedido do *Dispatcher*. Ele recebe um *script* contendo as informações sobre como executar determinada tarefa e a executa. Após o término, é responsável por enviar o resultado do processamento ao *Dispatcher* para que esse possa enviá-lo ao *Parametric Engine*.

3.3 Egg

Egg [53] é uma plataforma baseada em modelos econômicos para Grades Computacionais. Essa plataforma se apóia em alguns princípios:

- **Simplicidade e extensibilidade:** Toda operação a ser executada em uma Grade Computacional precisa envolver um pequeno número de funções padrão.

- **Alocação competente de recursos:** Os usuários não precisam especificar as máquinas que podem executar um determinado trabalho. São empregados mecanismos transparentes para descoberta e alocação de recursos.
- **Monitoramento e controle descentralizado:** Os participantes possuem mecanismos simples para influenciar o sistema como um todo. Organizações com recursos diferentes podem criar economias para compartilhar tais recursos mantendo o controle sobre eles.

Essa plataforma oferece a *Egg Shell* que é uma linguagem utilizada para descrever os requisitos de ambiente necessários para uma determinada aplicação, como também detalhes dessa aplicação. Através dela o usuário pode descrever o local de onde a aplicação deve ser copiada para que seja executada bem como possíveis dados de entrada.

São disponibilizadas *Caches* que representam os elementos funcionais como, por exemplo, computadores e dispositivos de armazenamento, entre outros. Dessa forma, quando um arquivo *Egg Shell* é transferido para uma *Cache*, essa o executa de acordo com seu propósito.

O processo de alocação de recursos é feito através do emprego de uma arquitetura econômica em que *Caches* competem entre si para executar uma determinada aplicação. Processos de leilão determinam o preço em função da demanda e definem um vencedor para utilizar um determinado recurso.

A plataforma permite que o usuário expresse os valores que pretende pagar em função do tempo de conclusão da execução. A variação do preço a ser pago em função do tempo de conclusão possui, inicialmente por motivos de simplicidade, um comportamento linear.

Cada *Cache* existente no sistema mantém uma métrica da taxa de sucesso no cumprimento dos tempos limite estipulados para as execuções. Dessa forma, é possível que o usuário especifique qual é o taxa mínima de segurança que uma *Cache* deve oferecer para que essa possa executar uma aplicação.

Essa plataforma permite que qualquer participante da Grade Computacional crie uma moeda a ser utilizada nos pagamentos de recursos e defina como deve ser a distribuição de

valores entre os participantes. No entanto, cada participante que fornece recursos pode definir quais os tipos de moedas são aceitos por ele. Sendo assim, cada tipo de moeda existente no sistema possui um banco responsável pelo armazenamento e segurança dos saldos dos participantes que utilizam tal moeda.

3.4 POPCORN

O *POPCORN* [54] é um sistema que oferece uma estrutura para construir um ambiente de computação global na *Internet*. Utilizando modelos econômicos para efetuar o compartilhamento de ciclos de UCP, *POPCORN* possui três tipos de entidades:

- ***Buyer:*** Entidade que consome ciclos de UCP compartilhados por um outro participante. É representada por um programa escrito em Java que utiliza o paradigma do *POPCORN* para expressar o paralelismo necessário.
- ***Seller:*** Entidade que disponibiliza seus ciclos de UCP para que possam ser utilizados por outros participantes (*buyers*).
- ***Market:*** Entidade que possui a função de intermediário entre um *buyer* e um *seller*.

Um programa que utiliza o paradigma do *POPCORN* para expressar seu paralelismo gera, para cada processamento remoto necessário, um *computelet* que é enviado a um *Market* (que tem um papel de intermediário). Um Market, por sua vez, envia um *computelet* para ser executado no *Seller* selecionado para tal operação. Após a execução, o *Seller* devolve o resultado ao *Market* que o encaminha ao *Buyer*.

POPCORN disponibiliza várias formas para definir o preço a ser cobrado por uma execução. É possível estabelecer o preço em função do número existentes de JOP'S, que são as operações Java a serem executadas. Uma outra maneira é estabelecer um preço para um *computelet* limitando seu tamanho através de ferramentas administrativas. Através de um leilão define-se qual *buye*r irá utilizar um determinado recurso.

Cada participante possui uma conta na qual é informada a quantidade de *popcoins* (moeda utilizada pelo *POPCORN*) que o participante possui. Antes de enviar um *computelet* para execução, o *Market* efetua, na conta do *buyer*, a dedução do valor a ser cobrado e, quando recebe o resultado do *Seller*, esse valor é somado em sua conta.

3.5 OurGrid

O *OurGrid* [68][69][70], desenvolvido pela Universidade Federal de Campina Grande, é uma ferramenta voltada ao compartilhamento de recursos em Grades Computacionais Ponto-a-Ponto, onde cada ponto representa um *site* (como, por exemplo, um conjunto de máquinas em único domínio administrativo). A Grade é concebida como uma rede de favores onde um participante, voluntariamente, oferece seus recursos ociosos para que outros possam utilizá-los e também utiliza recursos remotos quando sua capacidade local não é suficiente para uma determinada requisição.

Esse *middeware* possui como uma de suas características principais a execução de BoT´s (*Bag of Tasks*), as quais são tarefas que não possuem comunicação entre si e são bastante úteis em mineração de dados, processamento de imagens, quebra de chaves e simulações.

Como a participação na Grade é voluntária, para motivar a oferta de recursos nessa rede de favores, o *OurGrid* procura:

- Obter uma eqüidade, de maneira que o volume de recursos que um participante obteve da Grade seja proporcional àquele compartilhado por ele.
- Dar uma prioridade maior àqueles participantes que ofereceram mais recursos à rede.

Conforme ilustrado pela figura 3.3, os principais componentes do *OurGrid* são:

- *MyGrid*: O qual será melhor detalhado a seguir, faz o papel de intermediário utilizado pelo usuário no momento de interagir com a Grade Computacional.

- *OurGrid Community*: Responsável pela montagem da Grade a ser utilizada pelo componente *MyGrid*.
- *SWAN*: Responsável pelo acesso seguro aos recursos.

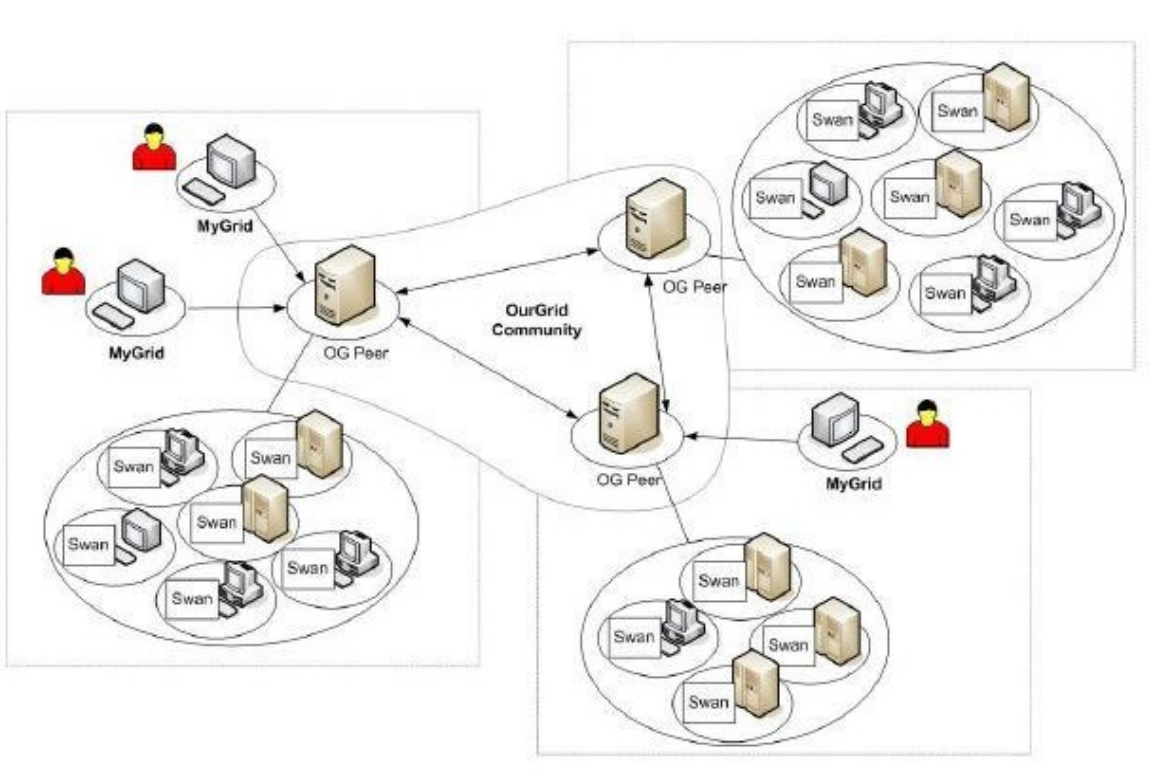


Figura 3.3: Principais componentes do OurGrid

3.5.1 MyGrid

O *MyGrid* [71] é responsável por efetuar a interface entre o usuário e a Grade Computacional. Quando esse usuário precisa executar uma aplicação parelela é necessário que ele tenha que se envolver o menos possível com detalhes da Grade. Sendo assim, um dos objetivos desse componente é simplificar os processos de instalação necessários para que o usuário possa fazer parte desse ambiente distribuído.

Para que o usuário possa executar uma aplicação na Grade é necessário que ele possua uma máquina onde ele coordene tal execução e outra onde a execução realmente ocorra. Essas máquinas não denominadas *home machine* e *grid machine*, respectivamente.

De maneira a obter a interoperabilidade necessária, os recursos disponíveis são representados através de interfaces. A GuM (*Grid Machine Interface*) é uma abstração para uma máquina, enquanto a GuMP (*Grid Machine Provider Interface*) representa um conjunto de máquinas, o qual é gerenciado por terceiros . Essas abstrações são bastante importantes visto que para se adicionar novos recursos à Grade como, por exemplo, uma máquina, basta implementar tal interface.

Conforme ilustrado na figura 3.4, quando se deseja utilizar recursos de um conjunto, o componente *scheduler* solicita os recursos ao GuMP, que após ganhar acesso a tais recursos devolve ao escalonador as interfaces (GuM's) para tais máquinas.

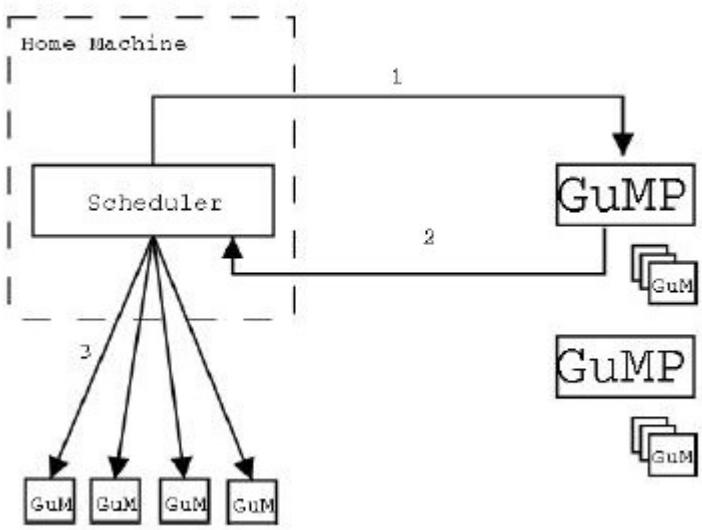


Figura 3.4: Arquitetura do MyGrid

O *MyGrid* oferece três implementações nativas para GuM's:

- *UserAgent:* Interface desenvolvida em Java que permite que sejam instaladas aplicações em uma *Grid Machine*.

- *GridScript:* Fornece as mesmas funcionalidades da *UserAgent*, porém na forma de *scripts*.
- *Globus:* Desenvolvido para fornecer interoperabilidade entre o *MyGrid* e o *Globus Toolkit*.

Capítulo 4

Plataformas de Apoio

Esse capítulo descreve algumas tecnologias e plataformas que apóiam o desenvolvimento e funcionamento de Grades Computacionais. Serviços *Web* são descritos na seção 4.1, esforços de padronização em 4.2 e *Globus Toolkit* em 4.3.

4.1 Arquitetura Orientada a Serviço

Um sistema distribuído é composto por um conjunto de elementos de *hardware* e *software* que se comunicam através da troca de mensagens [40]. A comunicação entre máquina e homem é algo já bastante difundido com a proliferação da *Web* e das tecnologias correlatas como, por exemplo, a linguagem HTML. No entanto a possibilidade de efetuar interações entre máquina-máquina, ou melhor, aplicação-aplicação vem ganhando um espaço considerável na área da computação. Algumas tecnologias já implementam mecanismos que possibilitam a execução de procedimentos remotos como, por exemplo, *Sun RPC* [40] e *Java RMI* [64]. Essas tecnologias possibilitam que uma aplicação invoque uma outra remotamente passando parâmetros de processamento e recebendo uma resposta como resultado de tal procedimento.

No entanto tais tecnologias são, em muitos casos, dependentes de plataformas e linguagens de programação, fato que as torna mais fechadas. As arquiteturas orientadas a serviços (*Service Oriented Architectures* - SOA) [43] fazem uso extensivo da tecnologia de serviços *Web* de maneira a definir uma estrutura orientada a serviços independente de plataforma e linguagem de programação.

4.1.1 Serviços Web

Um serviço *Web* [26,44] é um sistema de software projetado para fornecer formas de interação entre aplicações através de uma rede de computadores. Assim, uma determinada aplicação pode ser implementada na forma de um serviço e disponibilizada para ser invocada remotamente.

Analisando o contexto no qual os serviços *Web* são projetados, podemos encontrar alguns requisitos importantes:

- Diversas empresas e profissionais podem desenvolver serviços *Web* que implementam recursos e soluções para determinados assuntos. Sendo assim, é necessário que esses sejam publicados para, da forma mais simples possível, serem localizados e conseqüentemente utilizados;
- Os serviços *Web* apresentam uma interface pela qual aplicações-cliente podem interagir com esses serviços. Sendo assim, é necessário que a interface de serviços fornecidos esteja disponível para aplicações consumidoras;
- A interoperabilidade entre sistemas diferentes é uma questão bastante importante no âmbito de um sistema distribuído. Por isso, a troca de mensagens entre dois serviços deve ser estabelecida independente de plataforma ou linguagem de programação utilizada na construção do serviço;
- As mensagens trocadas entre as aplicações precisam, de alguma forma, serem transportadas entre a máquina origem e destino;

Para que a interoperabilidade seja mantida, diversas especificações e protocolos são utilizados. De uma maneira abstrata, podemos analisar o modelo empregado pelos serviços *Web* através da estrutura de camadas ilustrada na figura 4.1 [39].

Figura 4.1: Modelo de Camadas dos Serviços *Web*

O UDDI (*Universal Description, Discovery and Integration*) [44] fornece mecanismos para armazenamento de descrições de serviços e descoberta de serviços Web. Através de uma consulta a um repositório, é possível localizar serviços e obter informações que são necessárias para sua utilização. As informações sobre os serviços podem ser pesquisadas em [41]:

- **Páginas brancas:** As informações são fornecidas através dos nomes dos serviços e detalhes para contato;
- **Páginas amarelas:** As informações são fornecidas através dos tipos dos serviços ou das empresas;
- **Páginas verdes**: As informações são fornecidas através de detalhes técnicos como, por exemplo, tipo de transporte a ser utilizado;

Para que uma aplicação qualquer possa interagir com um serviço *Web* é necessário que essa interface seja fornecida de maneira que certas informações sejam conhecidas como, por exemplo, número e tipos dos parâmetros, tipo de retorno, entre outros. O WSDL [41, 44] é um documento em XML [46] (desenvolvido através de um consórcio mantido pela W3C) para descrever um serviço *Web* de maneira que uma outra aplicação possa trocar

informações com esse serviço. É possível efetuar uma analogia simples entre WSDL - serviços *Web* e IDL – *Corba* [45].

Compiladores foram desenvolvidos com a finalidade de analisar um documento WSDL e gerar um código que implemente *stubs* para o aplicativo cliente e *skeleton* para o servidor. *Stub*s e *skeletons* possuem a função de abstrair todo o processo de troca de mensagens entre a aplicação cliente e servidora, de forma que para o desenvolvedor seja fornecida uma maneira transparente de interagir com um serviço remoto.

É muito importante que a troca de mensagens entre duas aplicações possa ocorrer mesmo quando exista diferença entre plataformas e linguagens de programação utilizadas de maneira a permitir interoperabilidade. XML é uma linguagem padrão que vem se destacando muito no que diz respeito a troca de informações e independência de plataforma. Por isso o protocolo SOAP (*Simple Object Access Protocol*) [42,44] para troca de mensagens e invocação de procedimentos remotos é baseado em XML. Esse protocolo define como deve ser o formato de uma mensagem trocada entre duas aplicações e utiliza protocolos já existentes para o envio das mensagens como, por exemplo, HTTP, SMTP, entre outros [41].

Uma mensagem SOAP é organizada em um arquivo XML que representa, através de suas Tag's, um envelope (elemento raiz) contendo uma área de cabeçalho e uma área para o corpo da mensagem. Essa estrutura pode ser vista na figura 4.2.

O elemento *Header* é opcional e pode ser utilizado para funções específicas da aplicação no transporte da mensagem. Os intermediários através dos quais a mensagem passa entre a origem e o destino final podem processar esses cabeçalhos de maneira a implementar diversas funcionalidades como, por exemplo, autenticação, controle de transação, entre outros [42].

O elemento *Body* contém a mensagem propriamente dita que está sendo trocada entre as aplicações [41]. Esse corpo da mensagem pode implementar dois tipos de interação entre as aplicações:

- ***Document-Style*:** Nesse tipo de interação o elemento *Body* transporta uma mensagem XML que diz respeito a um documento que está sendo trocado entre as aplicações como, por exemplo, um pedido de compra contendo todos os itens, quantidade e preço. Dessa forma, o serviço destino recebe essa mensagem e processa esse documento.
- ***RPC-Style***: Nesse tipo de interação o elemento *Body* transporta um documento que carrega informações sobre uma chamada de procedimento remoto, contendo qual procedimento deve ser invocado e o valor de cada um dos seus parâmetros.

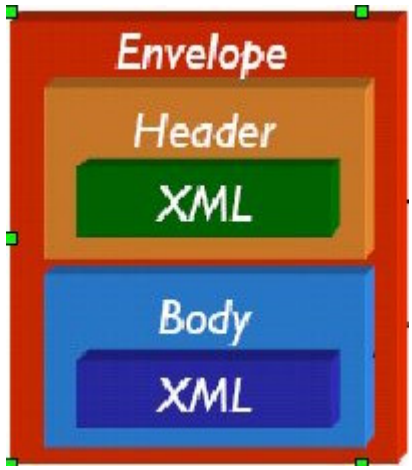


Figura 4.2: Mensagem SOAP.

Conforme ilustrado pela figura 4.3, o processo de utilização de um serviço *Web* inicia quando esse é desenvolvido e publicado em um registro UDDI . O interessado em utilizar tal serviço efetua uma busca no repositório e o encontra. O cliente, então, pode efetuar o mapeamento da interface do serviço através da descrição WSDL e finalmente interagir com o serviço *Web* através do protocolo SOAP.

Figura 4.3: Processo de utilização de um serviço Web.

A arquitetura básica de um ambiente de execução utilizando serviços *Web* e o protocolo HTTP no transporte das mensagens é mostrada na figura 4.4 [44]. No cliente são encontrados os seguintes componentes:

- **Implementação do cliente:** Aplicação cliente que irá invocar um serviço remoto;
- ***Stub*:** Módulo que é gerado pelo compilador através do processamento da interface do serviço *Web* descrito em *WSDL.* Tem o papel de abstrair a implementação necessária para a invocação do serviço remoto;
- **SOAP *Engine*:** Responsável pelo tratamento das mensagens SOAP. Pode estar acoplado ao aplicativo cliente;
- **HTTP *Engine*:** Responsável pelo transporte das mensagens entre os sistemas.

No servidor (provedor de serviços) são encontrados os seguintes componentes:

- **Implementação do servidor:** Aplicação que será executada em função de uma requisição efetuada pelo cliente;
- ***Skeleton*:** Possui um papel parecido com o *stub* existente no cliente, pois abstrai a implementação necessária para o tratamento da invocação, recebimento de parâmetros, entre outras funções.

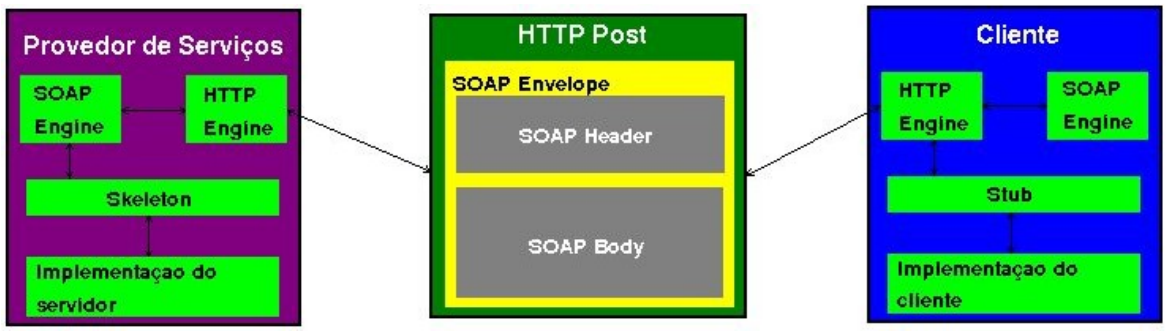


Figura 4.4: Arquitetura de um ambiente de execução utilizando serviços Web.

4.2 OGSA, OGSI e WSRF

Em um sistema distribuído a interoperabilidade é uma questão muito importante [10,47,48, 49]. Dessa forma, estabelecer um padrão a ser seguido pelas diversas instituições e desenvolvedores que produzem produtos para uma Grade Computacional contribui para que a interoperabilidade possa ser alcançada entre produtos distintos.

A OGSA (*Open Grid Services Architecture*) define que todos os recursos compartilhados são representados por serviços denominados Serviços de Grade, que são Serviços *Web* com interfaces bem definidas para atender certos requisitos como, por exemplo, descoberta e criação dinâmica de serviços, gerenciamento de tempo de vida, notificação, entre outros.

A OGSA define ainda requisitos referentes à identificação de instâncias de serviços, pois um mesmo serviço pode possuir diversas instâncias identificadas pelo *Grid Service Handle* (GSH). Um GSH é um nome globalmente único para uma determinada instância e não varia durante o tempo de vida do serviço. No entanto, um GSH não possui as informações necessárias para interagir com uma instância de um Serviço de Grade (como por exemplo, protocolo de transporte utilizado e endereço de rede), sendo assim, tais informações são armazenadas em um *Grid Service Reference* (GSR) que possui um tempo de expiração pré-determinado. Dessa forma, a OGSA define mecanismos para obter novos GSR's partindo de um determinado GSH.

A OGSA não descreve como um Serviço de Grade deve ser detalhadamente, pois ela descreve a arquitetura de uma Grade Computacional orientada a serviços de uma forma global com uma macro-visão. A OGSI (*Open Grid Service Infrastructure*) é uma especificação que descreve detalhadamente a arquitetura esboçada pela OGSA, definindo em detalhes como deve ser e se comportar um Serviço de Grade. A figura 4.5 ilustra essa relação entre a OGSA, OGSI e Serviços de Grade conforme a versão 3 do *Globus Toolkit* (implementação dessa última).

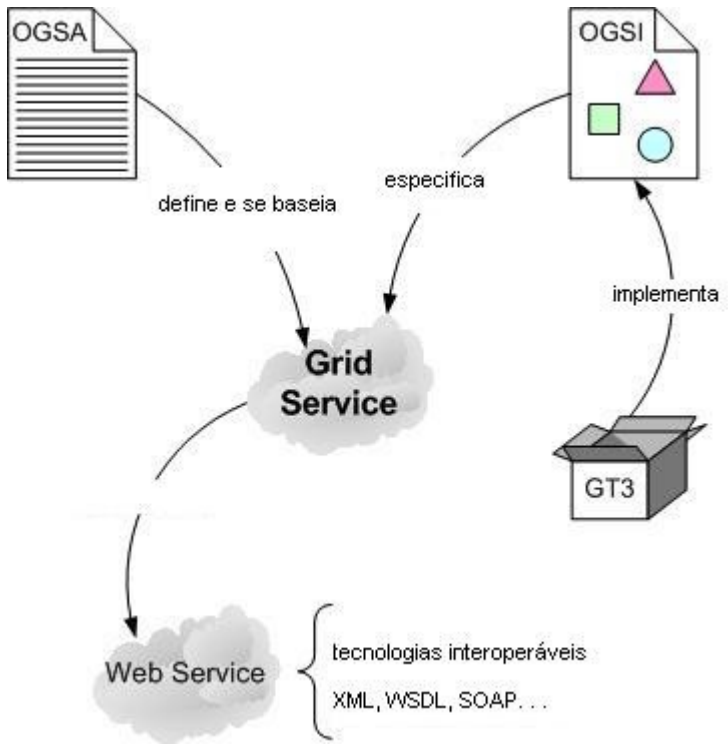


Figura 4.5: Relação entre OGSA e OGSI

De forma a efetuar uma simples analogia, é possível comparar a construção de um Serviço de Grade com a construção de uma casa. Pois o primeiro passo quando se deseja construir uma casa é procurar um arquiteto de maneira que esse efetue um projeto dando uma visão ampla de como a casa será depois de pronta. No entanto esse projeto não traz consigo detalhes de como essa casa deve ser feita. Isso exige que um engenheiro seja consultado de maneira a criar um projeto mais detalhado definindo como deve ser a estrutura dessa casa, a instalação elétrica e hidráulica, entre outros. Assim, na analogia

apresentada a OGSA faz um papel semelhante ao do arquiteto e a OGSI ao engenheiro [49].

Com a evolução da tecnologia de Serviços *Web*, alguns aspectos da OGSI precisaram ser analisados novamente [51]. Assim, surgiu a especificação WSRF (*Web Services Resource Framework)* [50,52] que refina a OGSI e divide a especificação em outras menores agrupadas em uma mesma família.

A necessidade de que seja possível que o estado entre diversas invocações de um Serviço *Web* seja mantido é um ponto bastante importante e discutido. De maneira a oferecer essa funcionalidade, a WSRF disponibiliza o *WS-Resource* que é definido como a composição de um Serviço *Web* e um recurso capaz de manter seu estado [52]. Tal recurso é expresso como um documento XML que pode ser criado, endereçado, acessado, monitorado e destruído através de mecanismos convencionais existentes na tecnologia de Serviços *Web*.

4.3 Globus Toolkit

O *Globus Toolkit* [22,23] é uma ferramenta de código aberto que teve sua primeira versão (1.0) lançada em 1998. Ela permite que recursos de processamento e armazenamento de dados, entre outros, sejam compartilhados de forma segura entre os participantes rompendo as barreiras de localização geográfica sem sacrificar a autonomia local.

Fornece um conjunto de programas, serviços e bibliotecas que auxiliam e possibilitam o desenvolvimento de Grades Computacionais, incluindo mecanismos de segurança, gerenciamento de recursos e dados, detecção de falhas e portabilidade.

Pelo fato de uma Grade Computacional ser, basicamente, um conjunto de aplicações distribuídas, a interoperabilidade entre elas é de fundamental importância. Dessa forma, o *Globus Toolkit 4* faz um uso extensivo da tecnologia de serviços *Web* para definir interfaces e estruturar seus componentes. A melhor forma para se garantir a interoperabilidade entre produtos desenvolvidos por entidades diferentes é a adoção de padrões. Ao observar a *Internet*, fica claro que uma das questões que a levaram a tal

patamar de adoção e sucesso é a padronização, pois para acessar uma página *Web* não é necessário que o navegador e o servidor *Web* sejam desenvolvidos por uma mesma empresa ou entidade. Dessa mesma forma, o *Globus Toolkit* adota especificações que ditam os padrões que devem ser seguidos de maneira a garantir tal interoperabilidade.

O *Globus Toolkit 4* utiliza as especificações da OGSA e do WSRF na implementação de seus serviços de maneira a adotar tais padrões conforme o modelo de camadas na figura 4.6.

O *Globus Toolkit 4* [23] é composto por um conjunto de módulos agrupados em cinco famílias. Essas famílias são: Segurança, Gerenciamento de Dados, Gerenciamento de Execução, Serviços de Informação e Módulo de Execução Comum. Ao analisar esses módulos, nota-se que a maior parte deles é implementada utilizando tecnologia de serviços *Web*. A figura 4.6 ilustra esses módulos e os identifica quanto à implementação baseada em serviços *Web* ou não.

O *Globus Toolkit 4* implementa um conjunto de serviços que caracterizam os módulos do servidor e disponibiliza também um conjunto de bibliotecas cliente [26].

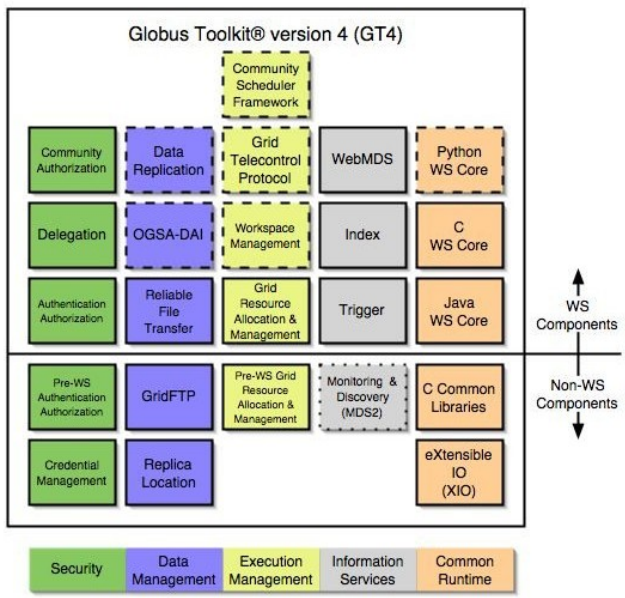


Figura 4.6: Módulos do *Globus Toolkit 4*

A figura 4.7 ilustra a arquitetura básica do *Globus Toolkit 4*. No servidor, os serviços pré-definidos podem ser agrupados em serviços *Web* ou não (do lado esquerdo e direito da figura respectivamente).

Os módulos do *Globus Toolkit* são descritos a seguir.

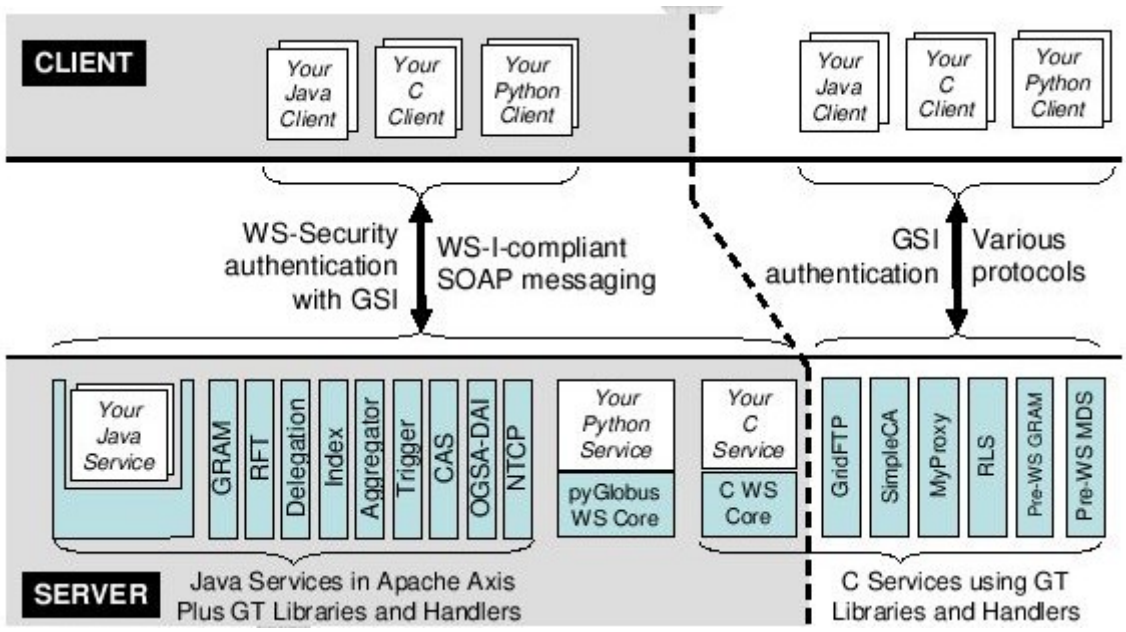


Figura 4.7: Arquitetura do *Globus Toolkit 4*

4.3.1 Módulo de Execução Comum

O *Globus Toolkit* implementa alguns componentes de maneira a disponibilizar ao programador os recursos necessários para o desenvolvimento de um serviço a ser executado em uma Grade Computacional [39]:

- ***Java WS Core**:* Disponibiliza recursos para o desenvolvimento de serviços utilizando a linguagem de programação *Java;*
- ***C WS Core*****:** Disponibiliza recursos para o desenvolvimento de serviços utilizando a linguagem de programação *C;*

- ***Python WS Core*:** Disponibiliza recursos para o desenvolvimento de serviços utilizando a linguagem *Python.*

Esses componentes podem ser combinados com outros elementos (como, por exemplo, *Web Servers*, SOAP *Engine*s, entre outros) para produzir os *Containers* utilizados para hospedar os serviços *Web* disponíveis. Um container pode ser utilizado para, além dos serviços criados pelo usuário, hospedar também os serviços do *Globus* (exemplo: GRAM, RFT, MDS) [26].

4.3.2 GRAM

Uma das principais funcionalidades de uma Grade Computacional é a execução remota de aplicações. Nem sempre a aplicação a ser executada e seus dados se encontram na máquina destino. É necessário que exista um controle de segurança no local onde a aplicação será executada, pois a aplicação pode ser maliciosa e causar danos ou prejuízos locais. Quando um cliente do GRAM (*Grid Resource Allocation and Management*) efetua uma submissão de um serviço para execução, esse pode especificar quais os arquivos devem ser transferidos para o servidor antes do início da execução (podendo ser o próprio executável do serviço e arquivos que serão lidos durante o processamento). O GRAM efetua essa transferência conhecida por *StageIn*. De forma semelhante, o serviço executado pode gerar arquivos contendo os resultados do processamento que são enviados ao cliente (*StageOut)*. O processo de *CleanUp* tem a responsabilidade de excluir os arquivos que foram gerados no servidor durante a execução.

É bastante importante que um serviço *Web* armazene seu estado. Os serviços Web que se encontram em um container do *Globus Toolkit 4* utilizam *WS-Resource* [55] para preservar seu estado. Para cada requisição de execução, o GRAM tem a responsabilidade de criar um *ManagedJob*, através do qual é possível obter informações sobre o serviço em execução. Após a criação, o GRAM retorna ao cliente o identificador (EPR) para o *ManagedJob*. Através desse identificador, o cliente pode efetuar chamadas ao GRAM de

maneira a requisitar informações sobre o estado do serviço. A tabela 4.1 ilustra os possíveis estados de um serviço.

Analisando os estados que um serviço pode possuir é possível analisar o ciclo de vida de um serviço através da figura 4.8.

Tabela 4.1:Possíveis estados de um serviço.

Estado	***Descrição***
Não submetido	Processo ainda não submetido para execução
StageIn	Processo aguardando pela transferência do executável e dos arquivos de entrada
Pendente	Processo ainda não iniciado pelo escalonador
Ativo	Em execução
Suspenso	Em execução, porém suspenso
StageOut	Execução completa. Transferindo os arquivos de resposta
CleanUp	Execução completa. Excluindo os arquivos
Concluído	Processo concluído
Falha	Processo falhou

Figura 4.8: Ciclo de vida de um serviço.

Conforme citado anteriormente, a segurança é uma questão que deve ser analisada e preservada em sistemas distribuídos. Para que a transferência de arquivos antes da execução (*StageIn*) possa ser efetuada, é necessário que o GRAM tenha autorização (credencial) para efetuar tal operação. Dessa forma, no momento da requisição de criação do serviço, o cliente deve enviar essa credencial para o GRAM. A utilização de uma

credencial também pode ser necessária para a execução do serviço, assim, de maneira semelhante à utilizada para o *StageIn*, essa credencial deve ser informada no momento da requisição.

Um determinado cliente utiliza o GRAM como uma interface com o servidor, de maneira a obter diversas funcionalidades necessárias durante uma execução remota de um serviço. No entanto, para prover muitas dessas funcionalidades, o GRAM precisa utilizar os serviços oferecidos por outros componentes. Conforme ilustrado na figura 4.9, o GRAM faz uso dos seguintes componentes:

- ***GridFTP*:** O GRAM, como dito anteriormente, é responsável por processos de transferência de arquivos antes e depois de uma execução. Dessa forma, o *GridFTP* é utilizado para realizar essa tarefa.

- **RFT (*Reliable File Transfer*):** Construído utilizando bibliotecas do *GridFTP*, o RFT herda características do *GridFTP* e acrescenta novos recursos com a finalidade de fornecer vantagens adicionais ao serviço como, por exemplo, confiabilidade de transferência.

- ***Delegation*:** Utilizado pelo cliente para delegar credenciais que sejam necessárias para a execução do serviço desejado.

- ***Local Scheduler*:** Para a execução propriamente dita de um serviço, o *GRAM* utiliza por padrão o mecanismo de *fork()* do *Unix*. No entanto, controles mais complexos de execução podem ser requeridos. Nesse caso, é necessária a utilização de um Escalonador Local (como por exemplo: *Condor, Nimrod/G*) que deve ser instalado e configurado.

- **GRAM *Adapter*:** É necessário quando um escalonador alternativo é utilizado. O GRAM *Adapter* é um *script* na linguagem *Perl* que mapeia as requisições de execução efetuadas pelo *GRAM* para o Escalonador Local apropriado.

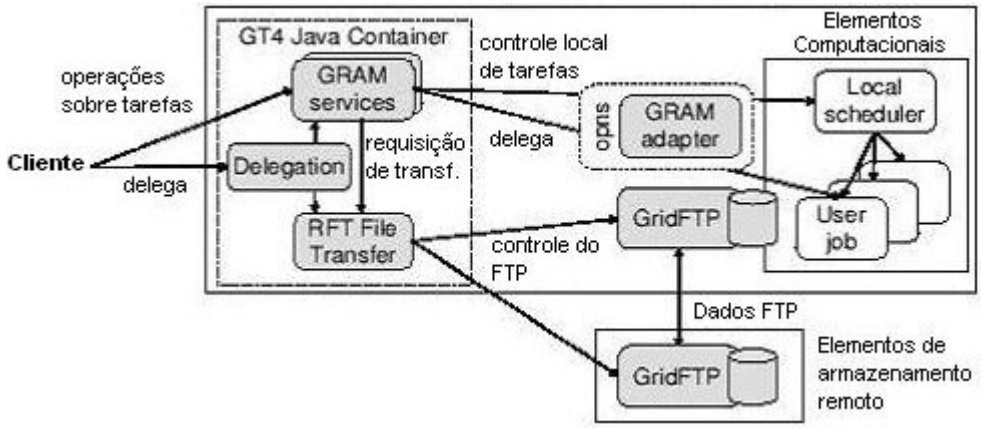


Figura 4.9: Arquitetura do GRAM

4.3.3 Gerenciamento de Dados

Em um ambiente de Grade Computacional, muitas aplicações envolvem manipulação de dados, transferências de arquivos da aplicação para o servidor (*StageIn*) e transferência de resultados da aplicação (*StageOut)*. Essas facilidades são realizadas pelo *GridFTP*, RFT (*Reliable File Transfer*), RLS (*Replica Location Service*) e OGSA-DAI (*Open Grid Services Architecture – Data Access and Integration*) descritos a seguir.

GridFTP é uma extensão do protocolo FTP para transferência de arquivos. Entre os motivos para se utilizar esse protocolo estão: sua grande utilização no mundo para transferência de arquivos, sua especificação aberta e sua arquitetura bem definida [31]. De maneira a atender aos requisitos referentes a gerenciamento de dados existente em uma Grade Computacional, o *GridFTP* implementa, entre outros, as seguintes funcionalidades [31]:

- **Apoio ao GSI *e Kerberos:*** São fornecidos os sistemas de autenticação GSI (*Grid Security Infrastructure*) e *Kerberos*.
- **Transferência paralela de dados:** De maneira a oferecer um serviço com melhor desempenho e a possibilidade de transferência paralela de dados, o *GridFTP* possibilita a utilização de múltiplos *streams TCP* durante uma transferência. Esses múlti-

plos *streams* podem ser utilizados em cenários em que um cliente se conecta a um servidor ou a vários servidores de maneira a aumentar o desempenho.

- **Transferência parcial de arquivos:** Em muitos casos não é necessária a transferência total de um arquivo para que o processo possa ser estabelecido. Dessa maneira, o *GridFTP* possibilita que apenas uma determinada parte do arquivo seja transferida.

Entre as desvantagens do *GridFTP* está o fato de não ser um serviço *Web* e não oferecer tratamento de falhas. Já o RFT é um serviço *Web* compatível com a especificação WSRF que utiliza as bibliotecas já existentes do *GridFTP* [30]. Dessa forma, ele herda as principais características desse componente [29] e agrega outras funcionalidades que se fazem necessárias. O RFT armazena o estado da transferência de forma persistente, de maneira que, se houver uma falha, é possível continuar a transferência do mesmo ponto onde foi interrompida.

O RLS auxilia na localização de réplicas de arquivos. Esse serviço emprega *Local Replica Catalog* (LRC) que mapeia nomes físicos para nomes lógicos e *Replica Location Index* (RLI). De maneira a fornecer balanceamento de carga e possibilitar que o sistema não tenha um ponto único de falha, o registro no LRC pode ser replicado. As réplicas podem ser localizadas através de consultas ao RLI [32].

A OGSA-DAÍ é um mecanismo para que recursos de dados (como, por exemplo, relacionais e XML) em uma Grade Computacional possam ser acessados por serviços *Web*. Dessa forma, serviços podem efetuar consultas e atualizações nesses dados [33].

4.3.4 Serviços de Informação

Uma Grade Computacional deve permitir o acesso a informações sobre a existência de determinados recursos e sobre o estado de um determinado recurso. Dessa forma, alguns componentes do *Globus Tookit* oferecem mecanismos que possibilitam:

- **Descoberta:** um recurso de uma Grade Computacional pode ser localizado em função de suas características técnicas e outros fatores como, por exemplo, o tempo no qual um serviço após ser submetido entrará em estado de execução.
- **Monitoração:** Em muitos casos é necessário que um determinado serviço ou recurso seja monitorado. Por exemplo, pode ser necessária a detecção de falta de espaço em um disco.

O MDS4 (*Monitoring and Discover System*) realiza essas tarefas. Esse sistema possui diversos serviços responsáveis pela coleta periódica e disponibilização de informações. Para isso, deve-se registrar com um *Aggregator Service*. Exemplos desses serviços são:

- ***Index*:** Responsável por colher informações sobre recursos e disponibilizá-los para consulta em um único local. Ele disponibiliza interfaces para consultas e assinaturas (quando se deseja receber notificações a respeito de um recurso). Um *Index Server* pode ser registrado em um outro índice de forma a se obter uma hierarquia de índices. Cada *Container* de serviço *Web* possui um *Index* instalado, de maneira que todos os serviços lá existentes sejam automaticamente registrados. É esperado que em uma organização virtual exista um índice através do qual seja possível localizar todos os recursos existentes [34, 35].
- ***Trigger*:** Em uma Grade Computacional muitos eventos podem ocorrer de forma inesperada e devem ser tratados apropriadamente. Por exemplo, o espaço em disco pode se tornar insuficiente durante a execução de um serviço que faz uso desse recurso. O *MDS-Trigger* monitora um determinado recurso ou serviço, coleta dados e compara esses dados com condições pré-estabelecidas. Se os dados atendem uma determinada condição, uma ação pré-determinada é executada (por exemplo: o envio de um *e-mail* ao administrador do sistema).

4.3.5 Segurança

As ferramentas de segurança preocupam-se em definir quem são os usuários, se realmente eles são quem dizem ser e quais operações eles são autorizados a fazer [26].

A GSI (*Globus Security Infrastructure*) é utilizada no *Globus Toolkit* para implementar os aspectos de segurança. Ela utiliza criptografia por chaves assimétricas como base de sua funcionalidade [38].

O Globus Toolkit 4 disponibiliza alguns componentes que possuem papéis bastante importantes em seu modelo de segurança:

- ***SimpleCA*:** É uma autoridade certificadora capaz de emitir certificados a usuários e serviços de uma Grade Computacional;
- ***Delegation*:** Disponibiliza uma interface para delegação de credenciais;
- ***MyProxy*:** É um repositório onde são armazenadas credenciais X.509.

Capítulo 5

Arquitetura para Grades Usando Modelos Econômicos

Esse capítulo apresenta uma arquitetura voltada ao gerenciamento de recursos através da utilização de modelos econômicos.

A arquitetura utiliza uma extensão do modelo de *Preço Afixado* e consiste em atribuir preços base aos recursos para diferentes tipos de dias e horários, pois a demanda pode possuir um comportamento variado. Assim, ajustando o preço em função desses fatores haverá uma maior distribuição da demanda, possibilitando que haja um possível equilíbrio entre a quantidade de recursos ofertados e a quantidade demandada.

A arquitetura aqui descrita não se limita a definir um preço estático para o recurso, pois diferentes requisições podem exigir diferentes esforços computacionais do recurso. Por isso, a arquitetura conta com um componente no consumidor e um no provedor que são responsáveis pela negociação de um recurso. Nesse processo de negociação o consumidor define (através da informação do tempo de UCP necessário para execução e do prazo no qual a conclusão deve ser realizada) o esforço computacional exigido. Assim, o provedor pode variar seu preço base em função do esforço exigido.

De maneira a estabelecer um cenário de concorrência, um consumidor que necessita de um recurso remoto solicita propostas a *n* provedores e utiliza aquele que melhor atenda seus objetivos.

Esse capítulo irá apresentar detalhadamente esses mecanismos e outros que também fazem parte da arquitetura como: o gerenciamento de Grupos de Trabalho para dar mais flexibilidade às organizações para manipular seus créditos, formas de pagamento, controle do comportamento honesto do participante, entre outros.

5.1 Grupo de Trabalho

Em uma grade computacional baseada em modelos econômicos são encontrados dois tipos de agentes: os provedores e os consumidores de recursos.

Para que um determinado participante possa fazer uso de um recurso da grade, é necessário que ele possua créditos suficientes para pagar ao provedor pela sua utilização. No entanto, para possuir créditos, é preciso que o participante ofereça recursos e esses sejam utilizados pelos demais que pagariam pelo seu uso.

Determinados problemas podem exigir um grande poder computacional com custo muito alto. Portanto, o valor de créditos obtidos por um certo participante pode não ser suficiente para pagar os recursos necessários. Uma solução para esse problema seria a criação de grupos de trabalho.

Um Grupo de Trabalho tem como objetivo principal reunir diversos participantes de maneira que todos os créditos recebidos por eles sejam tratados como um montante único do grupo. Dessa maneira, um certo participante do grupo que precise utilizar uma quantidade grande de recursos remotos tem disponível esse montante de créditos para pagar.

Uma determinada instituição pode decidir, em função de suas necessidades, como organizar seus participantes em Grupos de Trabalho ou manter suas máquinas trabalhando de maneira independente. Para a formação de um Grupo de Trabalho, é necessário definir os papéis dos participantes no grupo. Cada participante pode exercer um ou mais dos papéis abaixo:

- **Gerador de Créditos:** O participante tem a função de oferecer seus recursos para a grade de maneira a obter créditos para o Grupo de Trabalho.
- **Consumidor de Recursos:** O participante é autorizado a gastar créditos do grupo para utilizar recursos remotos. Em um mesmo grupo, podem existir vários Consumidores de Recursos.

- **Coordenador do Grupo:** O participante possui a função de coordenar o Grupo de Trabalho. Pode obter informações sobre os participantes do Grupo (recursos disponibilizados para a grade, política de preços, número de créditos produzidos, entre outros) e definir parâmetros. O coordenador do Grupo também tem a função de definir quanto cada um dos Consumidores de Recursos pode gastar.

A figura 5.1 mostra algumas instituições que participam de uma grade computacional. A instituição A possui uma formação com apenas um participante que pode consumir recursos da grade e os demais têm o papel de apenas produzir créditos para o Grupo. Na instituição C, todos os participantes possuem o papel de gerar créditos e também podem fazer uso de recursos remotos. Nesse grupo, o montante que cada um dos consumidores pode gastar é controlado pelo Coordenador de Grupo. Por fim, a instituição D não possui agrupamento algum e todos os participantes trabalham de forma independente.

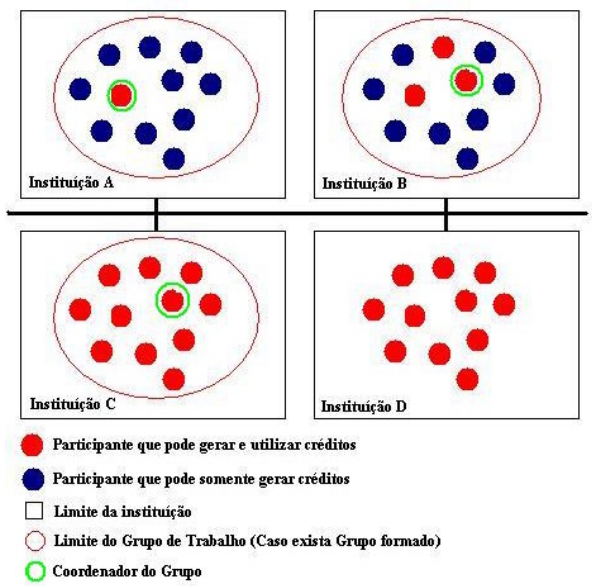


Figura 5.1: Exemplos de agrupamento de participantes

Os participantes e os Grupos de Trabalho de uma grade recebem identificadores fornecidos pelo componente GIS (*Grid Information Service*) que mantém informações sobre a grade (e será abordado na seção 5.2 a seguir). Os identificadores são:

- ***GGID**:* A *Grid Group ID* é uma identificação para um determinado Grupo de Trabalho.

- ***GPID:*** A *Grid Participant ID* é uma identificação para um determinado participante. Todo participante possui, além dessa identificação única, a identificação do Grupo de Trabalho ao qual ele pertence. Os participantes autônomos (que não pertencem a nenhum Grupo de Trabalho) também têm ambos identificadores (*GGID* e *GPID*), pois eles são tratados como um grupo de um único participante.

5.2 Arquitetura dos Participantes

A arquitetura aqui apresentada tem como base o *Globus Toolkit* e inclui novos componentes para o escalonamento baseado em modelos econômicos.

Conforme visto anteriormente, um mesmo participante pode desempenhar tanto o papel de gerador de créditos (provedor de recursos) quanto o papel de consumidor. Dependendo da organização dos participantes em Grupos de Trabalho, é possível que um determinado participante tenha ainda o papel de Coordenador de Grupo. Dessa maneira, dependendo dos papéis exercidos pelo participante, ele pode possuir todos os componentes aqui apresentados ou alguns deles.

Os componentes da arquitetura proposta (figura 5.2) são os seguintes:

- **GRAM [26,28]:** Componente oferecido pelo *Globus Toolkit* que possui, entre outras, a função de receber as requisições de execução remota. Possibilita as transferências de executáveis, arquivos de entrada para o processamento e de arquivos de saída (resultados de processamento).

- ***Scheduler Adapter* [26,28]:** Ao receber a requisição de uma execução, o GRAM precisa de um escalonador local que seja responsável pela execução do serviço. De maneira a oferecer maior flexibilidade quanto à utilização de escalonadores locais, o *Scheduler Adapter* é um *script* que tem a função de fazer a interface entre o GRAM e o escalonador em uso.

- ***Trader*:** Componente do provedor responsável por negociar com um consumidor, a utilização de um recurso local. Através dos dados e preços gerados pelo RPA (*Resource Price Agent*) para um determinado recurso, o *Trader* procura alocá-lo de forma a maximizar os lucros do provedor. Esse processo de negociação é estabelecido com o componente *Broker* de um consumidor interessado.

- **LLM:** O *Local Load Manager* é um serviço que permanece em execução enquanto o provedor permanece conectado à grade ou tempo integral (a critério do administrador do provedor). Com uma freqüência definida pelo administrador, o LLM verifica a carga do processador local e armazena essa informação no repositório de informações locais do provedor.

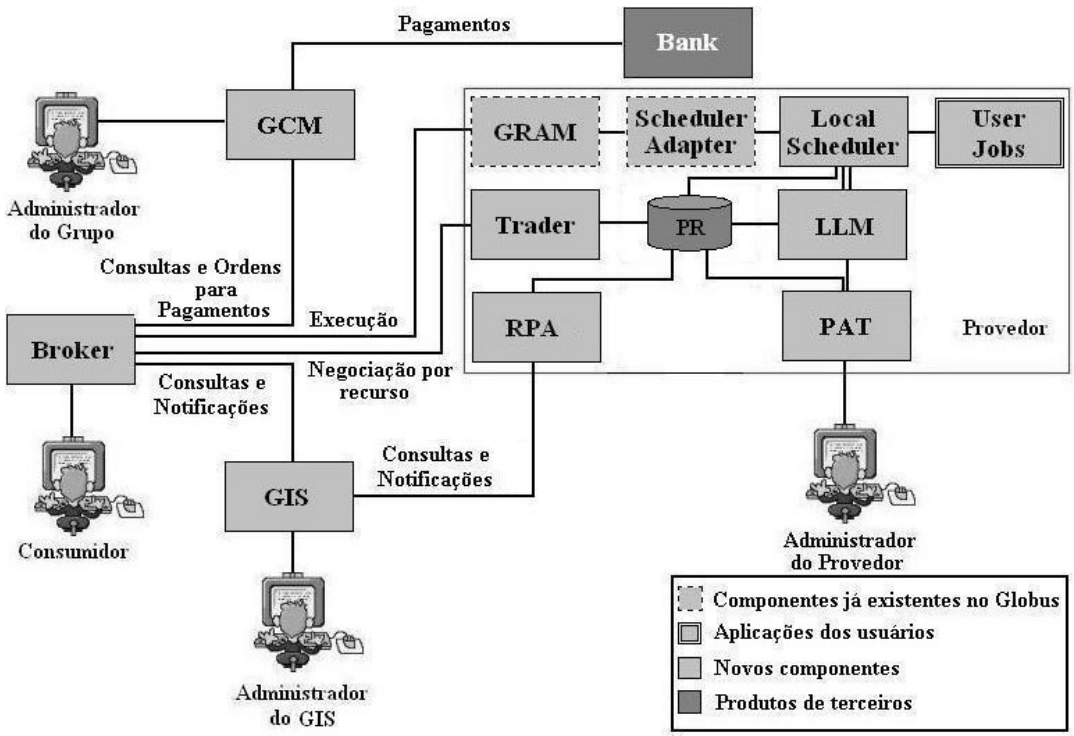


Figura 5.2: Arquitetura do sistema

- **RPA:** O componente *Resource Price Agent* (RPA) tem o papel de analisar, entre outros, os dados colhidos pelo LLM, a oferta e a demanda por um recurso (dados esses que são obtidos através de consultas ao GIS*)* e determinar, em função dos pa-

râmetros estabelecidos pelo administrador do provedor, os preços-base de um recurso.

- **GIS**: O componente *Grid Information Service* mantém um serviço de informações sobre recursos compartilhados de maneira que um consumidor interessado em utilizar um determinado recurso localize os provedores que potencialmente atendam suas necessidades. Ele mantém ainda um histórico sobre oferta, demanda, preço médio de recursos e reputação dos participantes (se o participante cumpriu o prazo estabelecido na negociação e, em caso negativo, devolveu os créditos cobrados).
- **GCM:** O componente *Group Credit Manager* coordena as operações financeiras de grupos. Possui a função de determinar quanto cada participante de um Grupo de Trabalho pode gastar, autoriza e efetua os pagamentos referentes ao uso de recursos por participantes do grupo.
- ***Local Scheduler*:** Escalonador local que tem a função de iniciar e gerenciar a execução de uma aplicação de um consumidor.
- ***Broker*:** Componente utilizado pelo consumidor no momento de submeter uma aplicação para execução. Considerando parâmetros estabelecidos pelo usuário (como por exemplo, limite de preço e prazo de execução), o *Broker* localiza e negocia com provedores de maneira a identificar aqueles que melhor atendam os critérios estabelecidos.
- ***Bank*:** Componente que controla a quantidade de créditos de cada participante (ou Grupo de Trabalho) e efetua as transações de pagamento referentes às utilizações de recursos.
- **PR:** O componente PR *(Provider Repository)* é o responsável pelo armazenamento persistente das informações locais de um participante. É implementado através da utilização de um SGBD distribuído por terceiros como, por exemplo, o *PostgreSQL* [67].

- **PAT**: O componente *PAT* (*Participant Administration Tool*) é uma ferramente Web que fornece os mecanismos e intefaces necessárias para que o administrador de um participante da grade computacional possa configurá-lo.

5.3 Determinação de Preços

As análises efetuadas nessa seção são voltadas aos mecanismos de geração de preços para recursos de processamento.

Para cada segundo de UCP é definido um preço-base que pode ser alterado em função do esforço computacional exigido para cumprir os prazos de execução.

No momento em que um provedor deseja compartilhar um recurso, o preço médio do recurso é estabelecido através de uma consulta a um GIS. Essa consulta auxilia o administrador a atribuir um preço ao recurso que seja condizente com a realidade na grade.

5.3.1 Determinação de Preços Conforme Padrões de Demanda e Carga

Em uma organização virtual, a demanda por um recurso, assim como a carga local de um provedor, pode variar em função de alguns fatores como dia da semana, dia do mês, período do dia, entre outros. Por exemplo, se certos participantes da grade forem empresas, é provável que as análises sobre os dados contábeis e gerenciais da instituição se intensifiquem, por motivos de fechamento, no último dia do mês.

Portanto, é interessante que o preço a ser cobrado pela utilização de um determinado recurso também varie conforme o dia ou horário. Dessa forma, um dia é dividido em períodos cuja duração é estabelecida pelo administrador. Os períodos do dia têm a mesma duração em minutos. Além disso, o administrador pode definir um dia como sendo do tipo:

- **Dia Normal:** Dia sem característica diferenciada;
- **Dia da Semana:** O administrador pode especificar preços diferentes em função de dias com demandas diferentes como, por exemplo, os domingos;

- **Dia do Mês:** Permite que sejam aplicados preços diferentes ao recurso em função de demandas diferentes em determinados dias do mês. No entanto, é possível que exista padrões de demanda tanto na ordem crescente dos dias do mês (como, por exemplo, dia primeiro, dia dez) como também na ordem decrescente (último dia do mês, penúltimo dia do mês, entre outros). Como os meses têm duração diferente, os dias de um mês são identificados, na forma decrescente, através de números crescentes iniciando por um (último dia do mês possui identificador 1, o penúltimo dia do mês possui o identificador 2, e assim sucessivamente).

Para que tais padrões sejam identificados é necessário que existam históricos a serem analisados. Assim, a cada negociação iniciada, o componente *Broker* associado ao consumidor, efetua uma busca no componente GIS pelo recurso desejado. Assim, o componente GIS mantém informações sobre a demanda por um recurso e efetua as análises necessárias para identificar os padrões de demanda. Quanto aos dados referentes à carga local, o componente LLM possui a função de monitorar o provedor e armazenar tais dados no PR. Dado um determinado período, os componentes RPA e GIS efetuam uma busca pelos históricos de carga e demanda respectivamente, para cada um dos tipos de dia. Por exemplo, para a análise da demanda nas segundas-feiras, o GIS busca no seu histórico qual foi a demanda registrada para o período nas *n* últimas segundas-feiras. O valor *n* é definido pelo administrador do provedor. Com base nos dados levantados, o componente GIS efetua o cálculo da média aritmética de demanda e do desvio padrão.

Para estabelecer a regra de identificação de padrões para um período, o administrador informa ao sistema qual é o percentual máximo do desvio padrão para que um período tenha seu comportamento considerado padrão. Para o cálculo da demanda média são consideradas as demandas dos últimos *n* dias. Para o cálculo do desvio padrão são consideradas a demanda média e as demandas dos últimos *n* dias.

5.3.2. Ajuste de Preços

O preço de um recurso pode influenciar na sua utilização já que consumidores tenderão a escolher, entre os recursos que atendem seus requisitos, os que tiverem o menor preço.

O RPA implementa um mecanismo automático de ajuste de preços que visa maximizar a taxa de utilização de recursos. Sendo assim, o preço deve ser proporcional à demanda e taxa de utilização, e inversamente proporcional à oferta.

Para ajustar o preço, o RPA analisa, para cada um dos registros de preço, o comportamento da oferta, demanda e da utilização do recurso (percentual de utilização da faixa disponibilizada para compartilhamento) nos diversos tipos de dias nos quais o período apresenta demanda padrão. Quando um recurso está sendo totalmente utilizado em um determinado período, sua taxa de utilização será cem por cento, o que causa um aumento do preço nesse período. Com esse aumento do preço, os consumidores que podem utilizar o recurso em outro horário tendem a procurar períodos mais baratos. Quando um recurso não está sendo totalmente utilizado, sua taxa de utilização é menor que cem por cento, o que causa uma queda do preço. Com um preço mais barato o período aumenta sua competitividade atraindo mais consumidores.

Através de consultas ao GIS, o RPA identifica se houve crescimento ou queda da demanda pelo recurso, o que faz com que o preço seja aumentado ou diminuído, respectivamente. De forma semelhante, o comportamento da oferta também é verificado, pois se ela aumenta, o preço tende a diminuir e, se essa oferta diminui, o preço aumenta. É considerada ainda a relação entre a oferta e a demanda, de maneira que se a oferta for maior que a demanda o preço diminui e se a demanda for maior que a oferta o preço aumenta.

A figura 5.3 mostra a fórmula para o cálculo do novo preço a ser cobrado em função de tais parâmetros, que variam ao longo do tempo (um período de tempo é representado por n). O preço de um recurso é denotado por p_n, a demanda, a oferta e a taxa de utilização são denotados por d_n, s_n e u_n, respectivamente. O administrador do provedor determina a proporção na qual ele deseja que os preços sejam alterados sob influência dos diversos fatores. Para isso, as variáveis a, β, χ, d e γ (que variam entre 0 e 1) definem o impacto da

demanda, oferta e relação oferta x demanda respectivamente. *d* e γ (que variam entre 0 e 1) estão relacionadas com o percentual de utilização.

$$p_n = p_{n-1}\left(1+\left(\frac{(d_n - d_{n-1})}{d_{n-1}} * \alpha\right) - \left(\frac{(s_n - s_{n-1})}{s_{n-1}} * \beta\right) + \left(\frac{(d_n - s_n)}{\min(d_n, s_n)} * \chi\right) - \left(\left(1 - \frac{u_{n-1}}{100}\right) * \delta\right) + \left\lfloor \frac{u_{n-1}}{100} \right\rfloor * \gamma\right)$$

Figura 5.3: Fórmula para o ajuste de preço

5.4 Negociação por um Recurso

Em uma grade baseada em modelos econômicos, é necessário que os consumidores e os provedores de recursos negociem de maneira a determinar o preço de um recurso.

Na arquitetura proposta, quando um consumidor deseja executar uma aplicação utilizando um recurso remoto ele invoca o componente *Broker*, o qual será responsável pela condução do processo. O *Broker* efetua uma consulta no componente GIS com a finalidade de encontrar os provedores que ofereçam o recurso necessário para tal requisição e, após obter a lista dos provedores encontrados, ele se conecta ao componente *Trader* de cada um dos provedores com a finalidade de estabelecer uma negociação para determinar o preço da utilização do recurso. A figura 5.4 ilustra a seqüência de interações entre os componentes no momento na negociação.

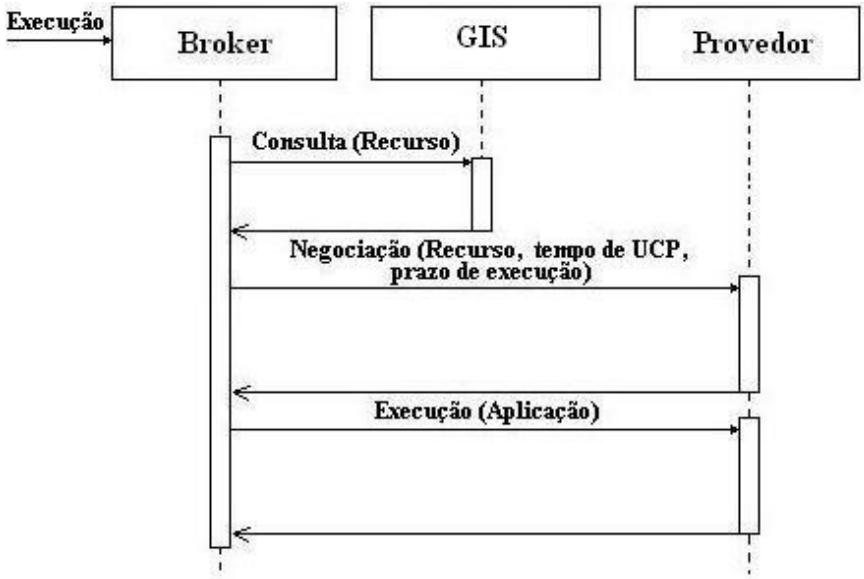


Figura 5.4: Diagrama de seqüência da negociação e execução.

5.4.1 Valor de um Recurso

Conforme visto anteriormente, o RPA atribui um preço-base a ser cobrado por segundo de UCP. No entanto, para prazos mais curtos, o preço deve aumentar. Por exemplo, dois clientes (A e B) que possuem, cada um, uma aplicação que consome 600 segundos de UCP, podem estipular como prazo limite para conclusão 1200 e 1800 segundos, respectivamente. Nesse caso, o cliente A exige que o provedor do recurso empenhe um esforço maior no processamento de seu trabalho em relação ao cliente B.

Um provedor de recursos pode estabelecer preços diferenciados em função do esforço computacional a ser empregado na execução da aplicação. A fórmula na figura 5.5 define o preço a ser cobrado por uma determinada execução. A fórmula considera o valor base a ser cobrado por segundo de UCP (p) e a quantidade de segundos de UCP necessários para executar o aplicativo (c). Como esse tempo de UCP varia em função da velocidade do processador, deve ser efetuada uma conversão em função do valor de *FLOPS* do consumidor ($f1$) e do provedor ($f2$). É calculado ainda um desconto proporcional ao percentual de UCP livre que a execução manterá (esse percentual é calculado em função do

tempo *r* limite para conclusão da execução). A variável η, conforme ilustrado na figura 5.6, determina o comportamento do preço em função do tempo limite para execução.

$$v = p * \frac{cf_1}{f_2} * \left(1 - \left(1 - \frac{\left(\frac{cf_1}{f_2} \right)}{r} \right) * \eta \right)$$

Figura 5.5: Fórmula para cálculo do valor a ser cobrado pela execução.

Quando η possui valor 1, o valor a ser cobrado tende a zero sem nunca atingir tal valor. No entanto, para η maior que 1 o valor a ser cobrado por tal execução pode ser zero ou até mesmo negativo (conforme exemplo apresentado na figura 5.6, quando η é igual a 1,5 e o prazo de execução é 15) em função do tempo limite para conclusão da execução. Sendo assim, para utilizar um mecanismo com uma queda do preço rápida (com η maior que 1) é necessário estabelecer um ponto r_{max} (calculado pela fórmula da figura 5.7) a partir do qual qualquer tempo limite para conclusão de execução não terá nenhum valor a ser cobrado.

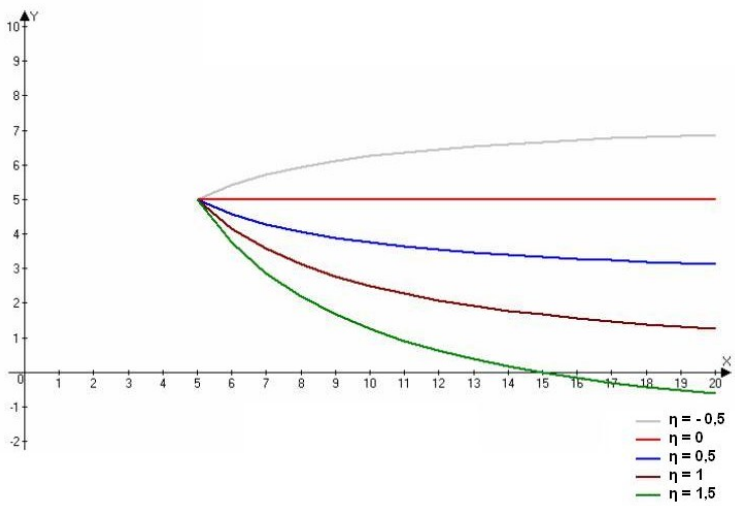


Figura 5.6: Influência da variável η no comportamento do preço

O valor da variável η influencia a alocação de recursos, pois ele pode incentivar os clientes a requisitarem prazos de execução mais longos. Para cada período do dia em um

dia específico, pode-se definir um valor para η dependendo do comportamento que se deseja obter.

$$r_{\max} = \frac{p\left(\frac{cf_1}{f_2}\right)^2 \eta}{p\left(\frac{cf_1}{f_2}\right)\eta - pc}$$

Figura 5.7: Fórmula para localizar o ponto a partir do qual o valor cobrado deve ser zero.

Com base na análise dos históricos de execução de um determinado período é possível observar qual é a necessidade dos clientes. É possível verificar se o percentual de UCP reservado para compartilhamento está sendo ocupado com execuções concorrentes de inúmeros processos com prazos longos de término (conforme figura 5.8) ou com execuções seqüenciais de processos com prazos mais curtos (confirme figura 5.9).

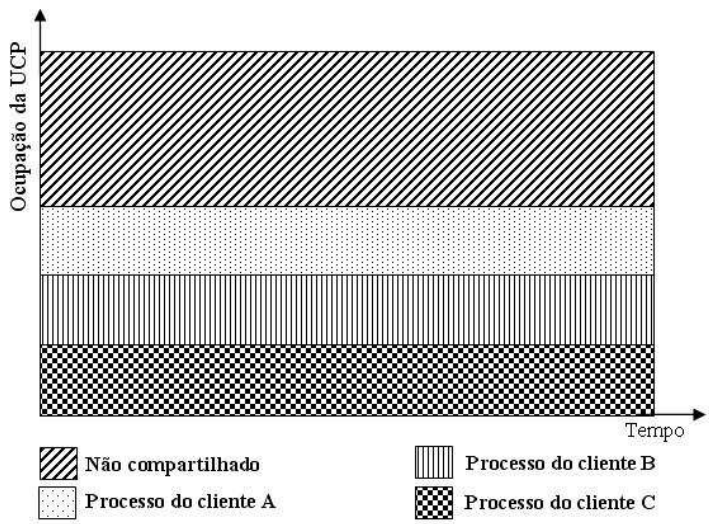


Figura 5.8: Ocupação da UCP por processos concorrentes

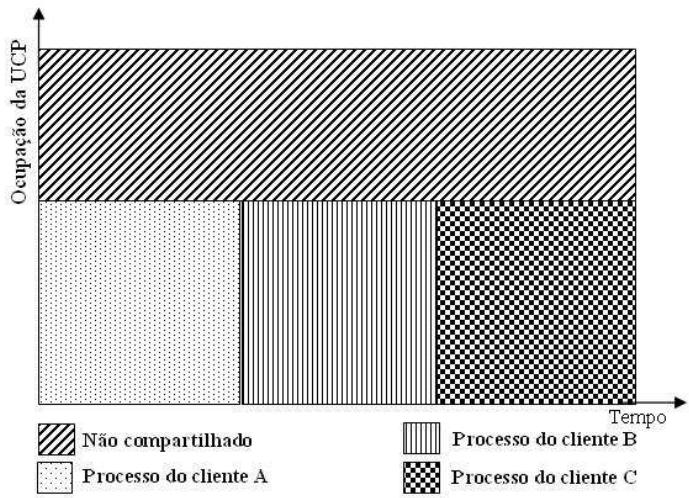


Figura 5.9: Organização em série dos processos na UCP.

Quando um cliente precisa do maior poder computacional possível para que seu problema seja resolvido em um menor espaço de tempo, é interessante que seja atribuída uma maior prioridade ao processo desse cliente. Assim, a variável η pode ser utilizada para tal finalidade, pois, se ela possuir um valor menor ou igual a zero os consumidores tendem a optar por uma execução com um menor limite de tempo para conclusão.

A arquitetura possibilita que a definição do valor de η seja efetuada pelo administrador ou de forma automática (através de processos de análise de históricos de execução).

5.4.2 Seleção de Recursos

O consumidor interessando em utilizar um recurso da grade tem por objetivo pagar o menor valor possível por esse recurso. Para isso é interessante que, no momento da negociação, vários provedores sejam consultados de maneira a localizar aquele que apresenta uma melhor oferta. Sendo assim, o consumidor precisa efetuar uma busca (em um GIS) por provedores que possuam o recurso desejado.

Após obter as identificações de todos os provedores com os quais deseja efetuar uma negociação, o consumidor inicia processos concorrentes de negociação.

Ao iniciar uma negociação com um provedor o consumidor informa quando deseja utilizar o recurso, o tempo de UCP necessário para a execução da aplicação (c), qual o valor de *FLOPS* (f_1) com o qual c foi calculado, qual é o limite máximo de tempo para conclusão (r) e o valor máximo que ele deseja pagar (z). Tendo essas informações, o provedor necessita definir qual é o intervalo de tempos nos quais a execução pode ser concluída (esse intervalo consiste nos tempos nos quais é possível executar a aplicação desejada considerando o preço que o usuário se dispõe a pagar e o prazo de execução) e informar ao consumidor o preço a ser cobrado para executar a aplicação em cada um desses tempos.

O processo de definição do intervalo de tempos (entre t_{mim} e t_{max}) depende do valor da variável η da fórmula na figura 5.5:

- **Para $\eta > 0$:** Nesse caso, o valor a ser cobrado diminui à medida que o tempo limite para conclusão aumenta. Assim, é possível calcular o tempo mínimo (t_1) de término da execução em função do preço máximo estabelecido pelo consumidor. Esse tempo pode ser calculado através da fórmula na figura 5.10 onde as variáveis possuem o mesmo significado daquelas na figura 5.5. No entanto, é necessário observar também qual é o tempo mínimo em que o recurso consegue concluir essa execução (t_2), pois o tempo de execução mínimo irá depender também da capacidade do recurso a ser utilizado. Sendo assim, o tempo mínimo utilizado na negociação é o maior dos dois tempos: $t_{mim} = max\ (t_1, t_2)$. Já o t_{max} (r) é estabelecido pelo usuário através do prazo de execução informado.

- **Para $\eta = 0$:** Nesse caso, o preço não possui variações em função do tempo limite para conclusão da execução. Sendo assim, o tempo mínimo (t_{mim}) é aquele no qual o recurso consegue concluir a execução da aplicação e o tempo máximo (t_{max}) é igual ao limite estabelecido pelo consumidor (r)

- **Para $\eta < 0$:** Nesse caso o preço aumenta à medida que o tempo limite para conclusão da execução aumenta. Dessa forma, o menor valor do intervalo (t_{min}) é o tempo

mínimo no qual o recurso consegue finalizar a execução. Já o tempo máximo é definido através da fórmula apresentada na figura 5.10.

$$t_1 = \frac{\left(\frac{cf_1}{f_2}\right)\eta}{\left(\frac{v}{p\left(\frac{cf_1}{f_2}\right)} + \eta - 1\right)}$$

Figura 5.10: Cálculo para determinar o tempo de execução em função do preço

Para cada um dos valores existentes no intervalo de tempos entre t_{min} e t_{max}, o provedor utiliza a fórmula na figura 5.5 para calcular o preço a ser cobrado para uma execução dentro desse limite de tempo. Assim, esses tempos e valores são inseridos em um documento XML (caso não tenha sido possível definir um intervalo de tempos, esse documento contém a representação de um intervalo vazio) que é enviado ao consumidor.

O consumidor ao receber as propostas efetua uma análise para definir qual deve ser o provedor utilizado na execução da aplicação. O usuário pode definir um critério de otimização a ser utilizado nessa análise, podendo esse critério ser:

- **Otimização de prazo de execução:** Nesse tipo de otimização é selecionado o provedor que termina a execução no menor tempo possível sem que o valor máximo a ser pago seja ultrapassado. Para isso, de cada proposta recebida é extraída uma *dupla* contendo o menor tempo no qual o provedor consegue concluir a execução e o valor a ser cobrado. As *duplas* extraídas dos provedores são ordenadas pelo prazo e, em seguida, pelo custo. Assim é selecionada a primeira proposta da lista cujo provedor ainda esteja disponível.

- **Otimização de custo:** É selecionado aquele provedor que forneça o serviço pelo menor preço sem que o tempo máximo para a execução seja ultrapassado. Para isso,

de cada proposta recebida é extraída uma *dupla* contendo o tempo no qual o provedor efetua o processamento cobrando o menor valor e o prazo. Essas *duplas* são ordenadas pelo valor cobrado e em seguida pelo prazo e é selecionada a primeira oferta cujo provedor ainda esteja disponível.

- **Otimização por *custo x benefício*:** Nesse tipo de otimização deseja-se utilizar aquele provedor que apresenta uma melhor relação entre o preço cobrado e o tempo limite. Para cada proposta recebida o consumidor calcula qual é o melhor custo benefício utilizando a fórmula na figura 5.11 (em que *v* representa o valor cobrado pela execução, *c* representa o tempo de UCP e *r* o limite de tempo para conclusão). Assim para cada proposta é gerada uma *tripla* contendo o *custo x benefício*, o valor e o prazo de execução. A forma de ordenação dessas *triplas* para definir qual provedor utilizar depende de um segundo critério de otimização que pode ser por prazo ou por valor, onde as *triplas* são ordenadas por *custo x benefício* seguido por prazo e valor ou ordenadas por *custo x benefício* seguido por valor e prazo, respectivamente.

$$b = \frac{v}{\left(\frac{c^2}{r}\right)}$$

Figura 5.11: Fórmula para calcular a relação entre custo e benefício

Após selecionar o provedor a ser utilizado na execução, o consumidor o notifica. No entanto, um provedor pode ter participado de vários processos de negociação diferentes de maneira que se muitos aceitarem suas propostas ele não será capaz de atender a todos. Assim, ao receber as mensagens dos consumidores que aceitaram as propostas, o provedor gera uma árvore de decisão contendo todas as possibilidades de execução (obedecendo sua capacidade de processamento) e seleciona a configuração que maximize seus lucros. Após

definir quais processos serão executados, o provedor envia mensagens de confirmação ou cancelamento da proposta aos consumidores.

No caso de cancelamento, o consumidor opta por utilizar a próxima proposta existente na lista.

A figura 5.12 ilustra o processo de negociação entre um consumidor e um provedor.

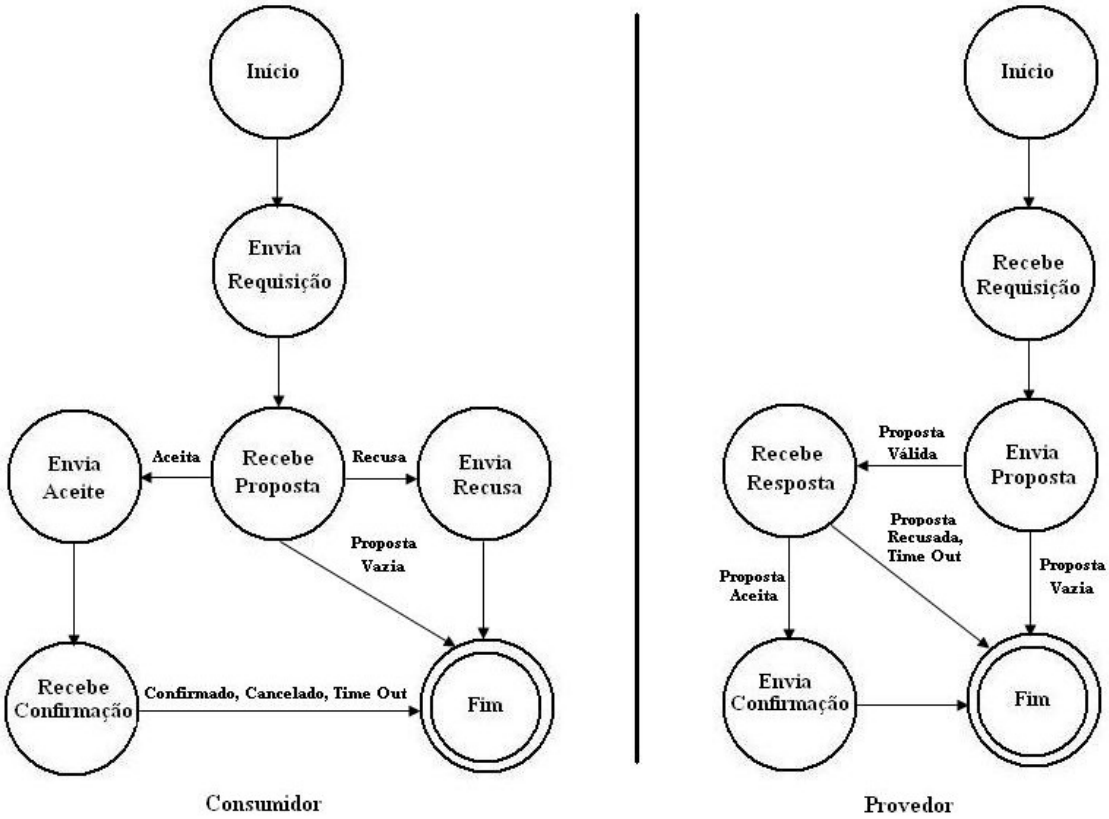


Figura 5.12: Máquinas de estados do processo de negociação entre consumidor e provedor.

5.5 Reserva Antecipada de Recursos

Um participante de uma grade pode ter uma previsão dos recursos que serão requisitados futuramente e informá-los ao *Broker*. Dessa forma, é possível que esse *Broker* efetue consultas junto aos provedores para ter uma estimativa do custo desses recursos. O custo é então informado ao coordenador do Grupo (ou o próprio participante caso não faça parte de nenhum grupo) que, baseado no saldo atual de créditos e no histórico de

compartilhamento de recursos, verifica se haverá créditos suficientes para pagar pelos recursos reservados.

5.5.1 Análise da Arrecadação de Créditos

A análise do histórico consiste em observar, para cada período existente entre o momento presente e o momento previsto para utilização do recurso, qual é o padrão de demanda por recursos locais e projetar a quantidade de créditos a ser obtida nesse intervalo.

Após essa análise, o usuário é notificado se será possível efetuar as alocações dos recursos remotos necessários. Em caso negativo, o administrador do Grupo pode tomar medidas para aumentar a arrecadação de créditos como, por exemplo, diminuir os preços dos recursos locais com a finalidade de aumentar suas taxas de utilização.

5.5.2 Reserva de Recursos

Para efetuar uma reserva de recurso, o *Broker* do consumidor deve obter orçamentos junto aos provedores. Na requisição de orçamento, deve ser informado o momento em que o recurso será utilizado. O provedor, ao reservar um recurso, deve manter o preço negociado no momento da reserva (mesmo que o preço base do recurso tenha sido alterado) até que o recurso seja utilizado.

O componente *Trader* do provedor deve considerar todas as reservas efetuadas para calcular o tempo mínimo de uma execução.

O provedor configura o percentual do valor negociado que deve ser pago no momento da reserva, valor esse que é perdido pelo consumidor caso não mais queira utilizar o recurso reservado. Sendo assim, no momento da execução o consumidor deve pagar apenas o valor restante.

5.6 Serviço de Informações

Através do GIS (*Grid Information Service*) o sistema fornece aos participantes informações referentes à grade computacional.

Os provedores registram seus recursos em um GIS de maneira que esses possam ser localizados pelos consumidores interessados em utilizá-los. No momento em que registra um recurso, o provedor deve selecionar de sua tabela o tipo ao qual tal recurso se enquadra. No momento da requisição de recurso, o consumidor deve informar qual o tipo de recurso desejado ou as faixas aceitáveis de determinadas características (como, por exemplo, processadores com memória *cache* entre 128 e 256 Kb).

O sistema fornece uma tabela dos possíveis tipos de processadores e os consumidores devem se referir a um tipo através de seu código nessa tabela. Registros de novos tipos podem ser feitos pelo administrador.

O GIS fornece ainda dados referentes à situação econômica da grade como, por exemplo, informações sobre oferta, demanda e preços. A demanda é registrada em um GIS no momento que um consumidor efetua uma busca, pois nesse momento ele precisa fornecer dados referentes ao recurso desejado, o tempo de UCP necessário e o valor de seu *FLOPS*. Quando um consumidor efetua uma consulta no GIS especificando o tipo do recurso desejado, a demanda é registrada exclusivamente para esse tipo. Quando a consulta é realizada por faixas, é criado um registro correspondente a essas faixas onde a demanda é registrada. Dessa forma, quando é necessário calcular a demanda por um determinado recurso, esse cálculo é realizado somando a demanda específica para o tipo de recurso à demanda por faixas.

Como uma organização virtual onde um determinado GIS opera tem participantes com diferentes velocidades de processamento, é necessário que o administrador do GIS defina um valor de *FLOPS* a ser utilizado nos registros de demanda por faixa. Dessa forma, esses registros são armazenados utilizando uma medida comum para todos os participantes.

Toda vez que uma negociação por um recurso é concluída com sucesso, o *Broker* (que trabalha associado a um consumidor) informa ao GIS qual foi a quantidade de segundos de

UCP adquiridos, seu preço e o tipo do recurso utilizado. Dessa forma, o serviço de informações mantém o preço médio negociado para um determinado tipo de processador. Para evitar que ocorram distorções na média, é utilizada uma ponderação em função da quantidade de segundos de UCP adquiridos.

Os dados de demanda, oferta e preço médio são armazenados em um histórico de maneira que seja possível a análise do comportamento da grade no decorrer do tempo.

Um GIS oferece informações referentes ao comportamento honesto dos participantes detalhadas a seguir.

5.7 Sistema de Pagamentos

A arquitetura proposta oferece um mecanismo para pagamento baseado no conceito de um banco virtual com a moeda G$. Cada Grupo de Trabalho (ou mesmo um participante autônomo) possui uma conta para controlar seu saldo. Essa conta fica armazenada no componente *Bank*.

Depois de uma negociação pela utilização de um determinado recurso, o consumidor deve efetuar uma ordem de pagamento junto ao componente *Bank* em benefício do provedor. Esse componente emite um recibo que é apresentado ao *Local Scheduer* do provedor para que esse tenha conhecimento que o valor negociado foi pago e inicie a execução da aplicação requisitada pelo consumidor.

São necessários mecanismos de segurança aplicados às transações de pagamentos para evitar possíveis ações fraudulentas. Para tal, quando um recibo é emitido por um banco ele contém as seguintes informações:

- **Número da transação:** Número único que representa a transação efetuada.
- **Número da negociação:** Número da negociação efetuada entre o provedor e o consumidor.
- **GGID do depositante:** Identificação do Grupo de Trabalho que efetuou o depósito.

- **GGID do favorecido:** Identificação do Grupo de Trabalho para o qual foi efetuado o crédito na conta.
- **Valor do depósito:** Valor debitado da conta do consumidor e creditado na conta do provedor.

Para garantir que o recibo foi realmente emitido pelo banco, ele deve ser autenticado utilizando a chave privada do banco. Dessa forma, ao receber esse recibo, o escalonador utiliza a chave pública desse banco para verificar se as informações referentes ao recibo são verdadeiras.

Esse mecanismo garante ao provedor o pagamento de um recurso que será utilizado. No entanto é possível que uma execução paga não tenha sucesso por motivos diversos como, por exemplo, provedor foi desligado durante a execução, queda do link de dados, consumidor não estava operante quando o resultado foi entregue, entre outros.

No processo de negociação, após o provedor enviar a confirmação para o consumidor notificando que sua aplicação será executada, um registro de estado de execução é armazenado no LLM do provedor. Esse registro possui as seguintes informações:

- **Número da negociação:** Número único no provedor que identifica o processo de negociação em questão.
- **Provedor ativo:** Informa se o provedor de serviços encontrava-se ativo no momento previsto para execução.
- **Execução iniciada:** Informa se a execução da aplicação do consumidor foi iniciada.
- **Execução concluída:** Informa se a execução da aplicação do consumidor foi concluída.

Quando um pedido de execução de um consumidor não é efetuado com sucesso pelo provedor é necessário que haja uma restituição do valor pago. Para isso o consumidor deve entrar em contato com o provedor para solicitar a restituição que deve ser feita ou não pelo provedor com base nas informações do registro de execução. A tabela 5.1 ilustra os possíveis valores para esse registro.

O administrador de um Grupo de Trabalho pode definir o critério de restituição automática de valores, pois é possível que sejam restituídas somente as operações com falha por culpa do provedor ou todas as operações não executadas. No entanto, todas as restituições solicitadas e não atendidas automaticamente pelo sistema são apresentadas ao administrador para que esse possa autorizá-las, manualmente.

Tabela 5.1: Possíveis valores do *Registro de Estado de Execução*.

Provedor Ativo	**Execução Iniciada**	**Execução Concluída**	**Restituir**
SIM	SIM	SIM	NÃO
SIM	SIM	NÃO	SIM
SIM	NÃO	NÃO	NÃO
NÃO	NÃO	NÃO	SIM

Quando uma execução não tem sucesso e o problema tiver ocorrido no provedor, esse deve restituir o valor pago. Caso, no momento do pedido de restituição, o provedor não esteja ativo é necessário aguardar um tempo e efetuar a solicitação novamente. Depois de um determinado número de tentativas, se o pedido de restituição for negado, o provedor armazena um registro de execução perdida no GIS. Esse registro possui as seguintes informações:

- **Número do registro:** Número único que identifica o registro.
- **GGID do provedor:** Número de identificação do Grupo de Trabalho do provedor.
- **GGID do consumidor:** Número do Grupo de Trabalho do consumidor que efetuou a reclamação.
- **Número da negociação:** Número da negociação pelo recurso estabelecida entre o provedor e o consumidor.
- **Valor:** Valor da operação.

No momento em que o GIS armazena um registro de execução perdida, é enviada uma notificação ao GCM do Grupo de Trabalho sob reclamação para que o administrador tenha ciência do fato ocorrido. Esse registro possui um tempo de vida, determinado pelo administrador, após o qual tal registro é removido do GIS. Um Grupo de Trabalho que possui um registro de execução perdida pode retirá-lo efetuando uma restituição do valor junto ao banco e apresentando o recibo ao GIS.

Quando é efetuada uma consulta por provedores de recursos, para cada provedor existente na resposta, o GIS fornece ao consumidor a relação de Grupos de Trabalho que tiveram reclamações. Dessa forma, o *Broker* pode optar por não utilizar provedores que possuam um número maior que *n* de reclamações.

Capítulo 6

Detalhes de Implementação

Esse capítulo apresenta as ferramentas utilizadas na implementação da arquitetura proposta na seção 6.1 e mais detalhes sobre os componentes na seção 6.2.

6.1 Ferramentas e Plataformas

A heterogeneidade encontrada em uma grade computacional requer que os produtos desenvolvidos sejam portáveis entre diversas plataformas. Dessa forma, a arquitetura descrita no capítulo anterior foi implementada fazendo uso das tecnologias *Java* [55,56]. Para tanto, foi adotado como ambiente de desenvolvimento o *NetBeans* 5.0 [57].

A necessidade de comunicação entre os componentes de diferentes participantes motivaram a utilização de serviços *Web* [44] para tal finalidade. Sendo assim, a interação entre os participantes ocorre com as vantagens de independência de plataforma, possibilitando futuras extensões da arquitetura e desenvolvimento utilizando outras linguagens de programação. A implementação da arquitetura proposta utiliza *Apache Axis* [58] como implementação do SOAP sendo hospedado no *Apache TomCat* [59].

O desenvolvimento de produtos com interface *Web* como, por exemplo, o componente PAT (*Participant Administration Tool*) foi realizado utilizando a tecnologia JSP (*Java Server Pages*) [56], fazendo uso do próprio *TomCat* como *Servlet Container*.

Quanto ao sistema operacional, optou-se em utilizar um sistema de uso livre e que oferecesse os recursos necessários. Dessa forma, o Fedora *Linux* [60] foi escolhido por atender de maneira bastante satisfatória a todos os requisitos.

6.2 Detalhamento dos Componentes

Essa seção detalha alguns dos componentes da arquitetura que foram total ou parcialmente implementados.

6.2.1 Local Scheduler

O componente *Local Scheduler* que é responsável pela execução de uma aplicação em um determinado provedor foi o único componente desenvolvido na linguagem C [61] já que essa linguagem oferece maior facilidade de utilizar recursos do sistema operacional e possui um melhor desempenho.

Como a qualidade de serviço (QoS) é um dos requisitos atendidos na arquitetura aqui descrita, é necessário que o *Local Scheduler* seja capaz de concluir a execução de uma aplicação dentro do prazo estipulado. Para isso, ao receber uma aplicação a ser executada, esse componente recebe também o tempo de UCP necessário para tal processamento e o prazo de execução.

O escalonador no momento da execução da aplicação efetua uma operação de *fork* [65] criando um processo filho que, ao invés de possuir um código executável igual ao do pai, recebe o código executável da aplicação requisitada. Dessa forma, o escalonador tem conhecimento do número do PID (*Process ID*) do novo processo criado e o monitora com relação ao percentual de tempo de UCP concluído e ao prazo de execução. Se for detectado que o processador não conseguirá concluir dentro do prazo, o escalonador altera a prioridade do processo ou até mesmo o algoritmo de escalonamento para conseguir terminar a tarefa dentro do prazo.

6.2.2 *Local Load Manager*

Tendo como responsabilidade o monitoramento da UCP de um provedor de recursos, o *Local Load Manager* (LLM) é executado como um serviço no *Linux* e verifica constantemente o percentual que o processador fica ocioso e o percentual de tempo que

esse processador está empregado no atendimento de requisições de consumidores da grade computacional. Para obter tais informações, o LLM faz uso dos arquivos encontrados no diretório "/proc" [66] do Linux.

O LLM, além de monitorar as dados citados acima, tem a função de fornecer informações ao componente PAT com relação às aplicações de consumidores que estão em execução em um determinado momento. Para tal, toda vez que o *Local Scheduler* inicia uma nova aplicação, ele se comunica com o LLM (através de *sockets* [62]) e informa o número do PID do processo. A partir desse momento, o LLM monitora esse processo até que esse seja concluído. Sendo assim, quando o PAT necessita colher informações sobre as aplicações em execução, ele se conecta ao LLM e obtém tais informações para apresentar ao usuário.

6.2.3 Resource Price Agent

O componente RPA (*Resource Price Agent*) é executado como um serviço no *Linux* permanecendo ativo durante todo o tempo para efetuar as análises dos dados periodicamente. O RPA busca padrões na utilização dos recursos para auxiliar o administrador do provedor a identificar os períodos com determinados comportamentos.

Quando o usuário, através do componente PAT, solicita uma análise imediata dos padrões de utilização do recurso, o PAT utiliza o RPA para gerar tais informações a serem apresentadas ao usuário.

Como um dos papéis do RPA é informar ao GIS dados locais (como, por exemplo, a oferta de recursos), toda vez que esse serviço é iniciado ele transmite, por meio de um serviço *Web*, ao GIS um arquivo XML contendo as informações referentes aos dias anteriores, cujos dados ainda não tenham sido transmitidos.

Para que os preços dos recursos locais possam ser reajustados automaticamente, o RPA necessita obter informações globais a respeito do estado da "economia" da grade computacional. Para isso, através de um serviço *Web*, o RPA efetua consultas no GIS e obtém esses dados.

6.2.4 *Trader*

O componente *Trader,* que é responsável pela negociação de um recurso entre o provedor e o consumidor, é dividido em dois módulos:

- Módulo de Comunicação: Implementado como serviços Web, esse módulo é responsável por receber as requisições e respostas dos consumidores e enviá-las ao Módulo de Análise.
- Módulo de Análise: Esse módulo é executado como um serviço no *Linux* e centraliza o processamento de todas as requisições recebidas dos clientes. Pelo fato de todas as requisições serem centralizadas em um único processo, é possível que, ao efetuar determinadas análises, todas as negociações em andamento possam ser consideradas em conjunto para maximizar os lucros do provedor.

Dessa forma, através da invocação dos serviços Web, o componente *Broker* do consumidor envia requisições ao Trader que, por meio de acessos ao banco de dados, determina as políticas de negociação estabelecidas pelo administrador do provedor e os preços a serem cobrados.

6.2.5 GIS

O componente GIS (*Grid Information Service*) é, atualmente, executado em um único participante da grade computacional coletando e fornecendo informações sobre a economia. As informações coletadas pelo GIS são armazenadas de forma persistente em um banco de dados local.

Para a interação entre o GIS e os demais componentes, foram desenvolvidos serviços *Web* que são responsáveis pelo recebimento e fornecimento de informações junto aos participantes da grade computacional. Dessa forma, através de invocações a esses serviços, o componente RPA de um provedor pode consultar informações como, por exemplo, demanda e oferta por um tipo de recursos, como também fornecer as informações locais de oferta ao GIS. De forma semelhante, o componente *Broker* de um consumidor invoca um

serviço *Web* para receber a lista de provedores que fornecem um determinado recurso e, no momento dessa consulta, o Broker atualiza os dados referentes à demanda por um recurso.

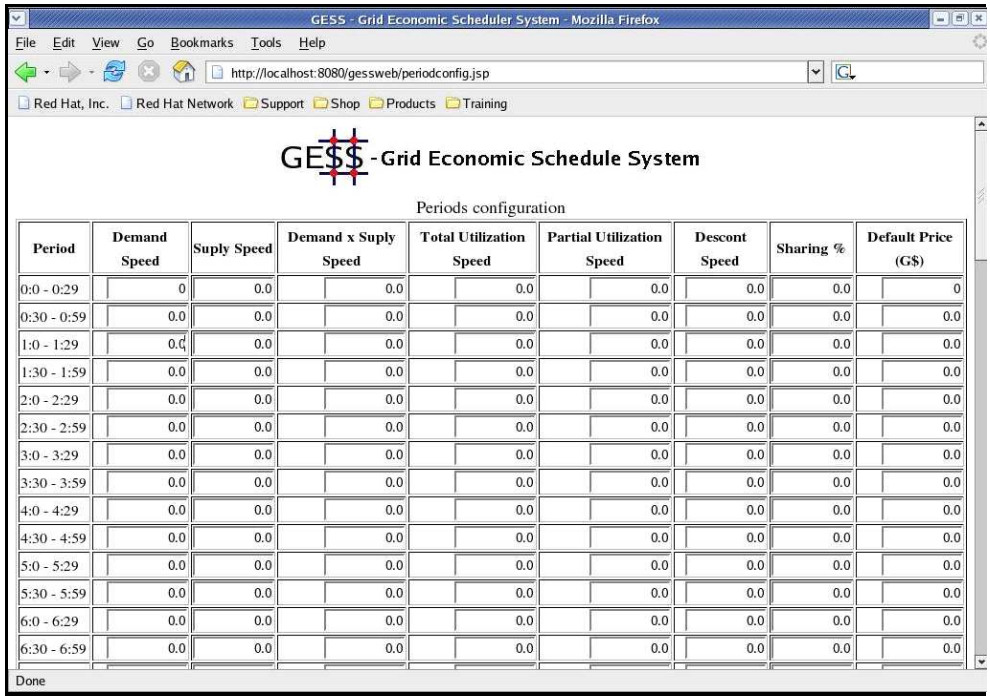
6.2.6 *Participant Administration Tool*

Através da utilização da tecnologia JSP (*Java Server Pages*) foram criadas aplicações *Web* que auxiliam a administração de um participante. Através dessas aplicações o administrador pode definir todos os parâmetros necessários.

Definição de Parâmetros de Negociação

De maneira a flexibilizar a política de negociação de um participante, o administrador pode definir parâmetros (como, por exemplo, velocidade de aumento do preço em função da oferta e demanda, taxa de diminuição do preço em função do prazo de execução, etc). É possível definir parâmetros para os vários períodos diferentes do dia. Esses parâmetros podem ser definidos nos seguintes níveis:

- Período: São informados, para cada período, os parâmetros de negociação e um preço padrão. A figura 6.1 ilustra a interface de administração dos parâmetros por período.
- Por tipo de dia: Para cada tipo de dia (como, por exemplo, segunda-feira, primeiro dia do mês, último dia do mês) são definidos os parâmetros de negociação. A figura 6.2 apresenta a interface que implementa essa opção.



Period	Demand Speed	Suply Speed	Demand x Suply Speed	Total Utilization Speed	Partial Utilization Speed	Descont Speed	Sharing %	Default Price (G$)
0:0 - 0:29	0	0.0	0.0	0.0	0.0	0.0	0.0	0
0:30 - 0:59	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1:0 - 1:29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1:30 - 1:59	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2:0 - 2:29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2:30 - 2:59	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3:0 - 3:29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3:30 - 3:59	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4:0 - 4:29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4:30 - 4:59	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5:0 - 5:29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5:30 - 5:59	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6:0 - 6:29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6:30 - 6:59	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

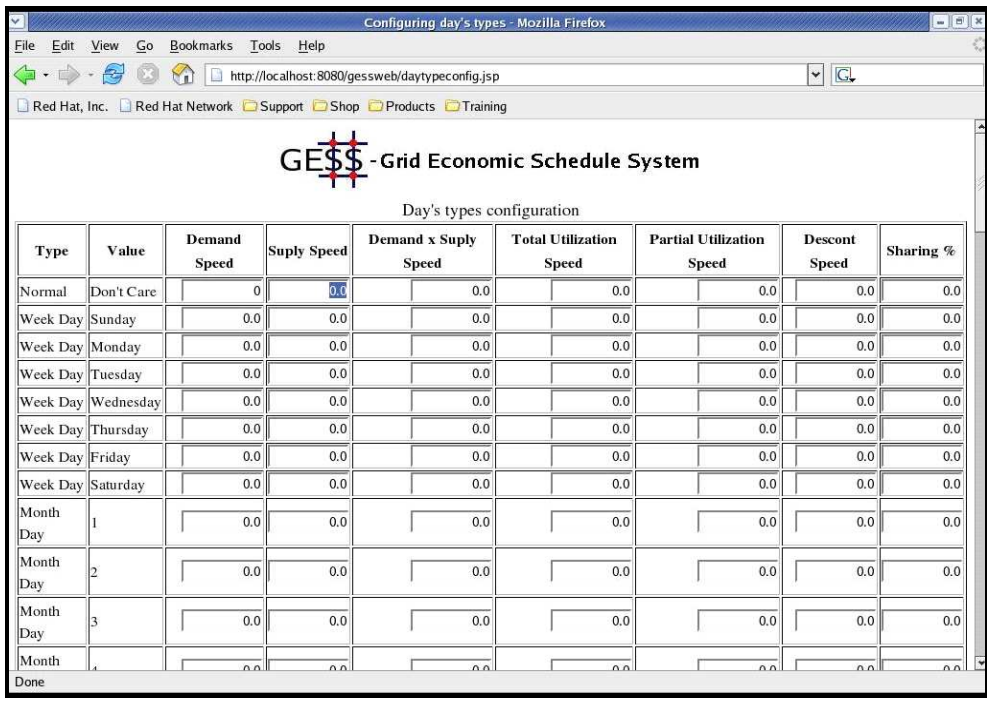
Figura 6.1: Interface de administração de parâmetros para os períodos.

Determinação de Preços

Os preços base de um recurso para um determinado período podem ser definidos manualmente pelo administrador do provedor e, a partir desse momento, o componente RPA ajusta-os automaticamente de acordo com os parâmetros definidos.

De maneira a oferecer ao administrador a possibilidade de disponibilizar um mesmo recurso com preços diferentes em função de determinadas situações como, por exemplo, finais de semana e início do mês, o mecanismo de preços admite que um mesmo recurso possua, para um mesmo período, diferentes preços em função do tipo do dia. Dessa forma, no momento de negociar com um consumidor o componente *Trader* busca o preço que se enquadre ao tipo do dia atual. Para o caso de dias que se enquadrem em mais de um tipo de dia (pois um mesmo dia pode ser domingo e início de mês, por exemplo), o administrador define uma ordem de prioridade a ser obedecida pelos componentes *Trader*.

O componente RPA também necessita dos parâmetros de reajuste para recalcular o preço periodicamente.



Type	Value	Demand Speed	Suply Speed	Demand x Suply Speed	Total Utilization Speed	Partial Utilization Speed	Descont Speed	Sharing %
Normal	Don't Care	0	0.0	0.0	0.0	0.0	0.0	0.0
Week Day	Sunday	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Week Day	Monday	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Week Day	Tuesday	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Week Day	Wednesday	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Week Day	Thursday	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Week Day	Friday	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Week Day	Saturday	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Month Day	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Month Day	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Month Day	3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Month								

Figura 6.2: Interface de administração de parâmetros para os tipos de dia

Ao definir um preço para um período, o administrador seleciona de onde (período, tipo do dia ou registro de preços) ele deseja que o parâmetro de negociação (política de cálculo de valor a ser cobrado pela execução) e reajuste de preços (velocidade de ajuste de preços em função da oferta, demanda e taxa de utilização, entre outros) sejam lidos. Dessa forma, o administrador pode definir uma parametrização específica para o caso em questão no próprio registro de preços. Ou ainda, definir uma parametrização mais genérica através para o período ou para o tipo do dia.

A figura 6.3 apresenta a interface de administração de preços para os períodos. Através do *Combo Box Behavior Type*, o administrador define onde o Trader e o RPA devem buscar os parâmetros.

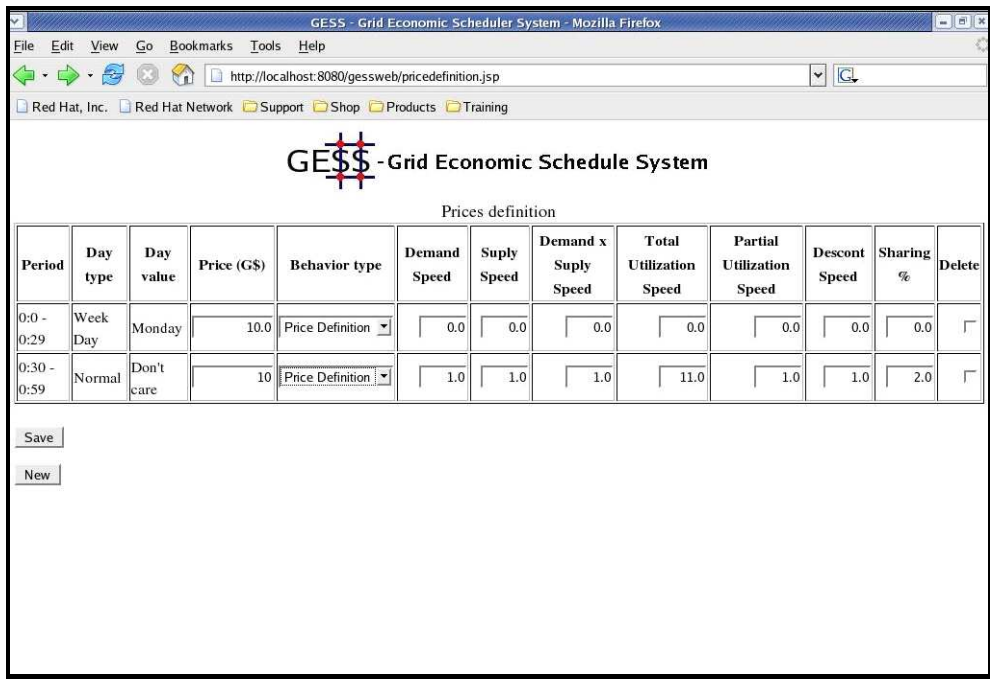


Figura 6.3: Interface para administração de preços

Análise de Padrões de Utilização da UCP

O componente RPA possui a função de analisar os dados gerados pelo LLM em busca de padrões de utilização do recurso de processamento. Esse processo é executado com uma freqüência pré-estabelecida e necessita dos seguintes parâmetros:

- Número de dias a serem analisados: Define a quantidade de dias que devem ser obtidos do histórico para serem analisados em busca de padrões.
- Limite do mês a ser analisado: Esse parâmetro é utilizado na busca por padrões de utilização em função do dia do mês. Assim, através desse valor o administrador informa até qual dia do mês, na ordem crescente, ele deseja que os dias sejam analisados.
- Limite do mês a ser analisado na ordem decrescente (ou reversa): Possui a mesma função do parâmetro anterior, no entanto é aplicado nas análises em ordem descrescente.

- Diferença máxima entre o desvio padrão e a média: Define qual deve ser o percentual máximo de diferença entre a média e o desvio padrão encontrados durante a análise da amostra para que possa ser considerado um comportamento padrão.

Tamanho mínimo da amostra: Define qual deve ser o número mínimo de ocorrências encontrados no histórico para que a análise possa ser considerada válida.

O PAT fornece uma interface, conforme ilustrado na figura 6.4, para a definição desses parâmetros e para possibilitar que o administrador inicie um processo de análise de padrões.

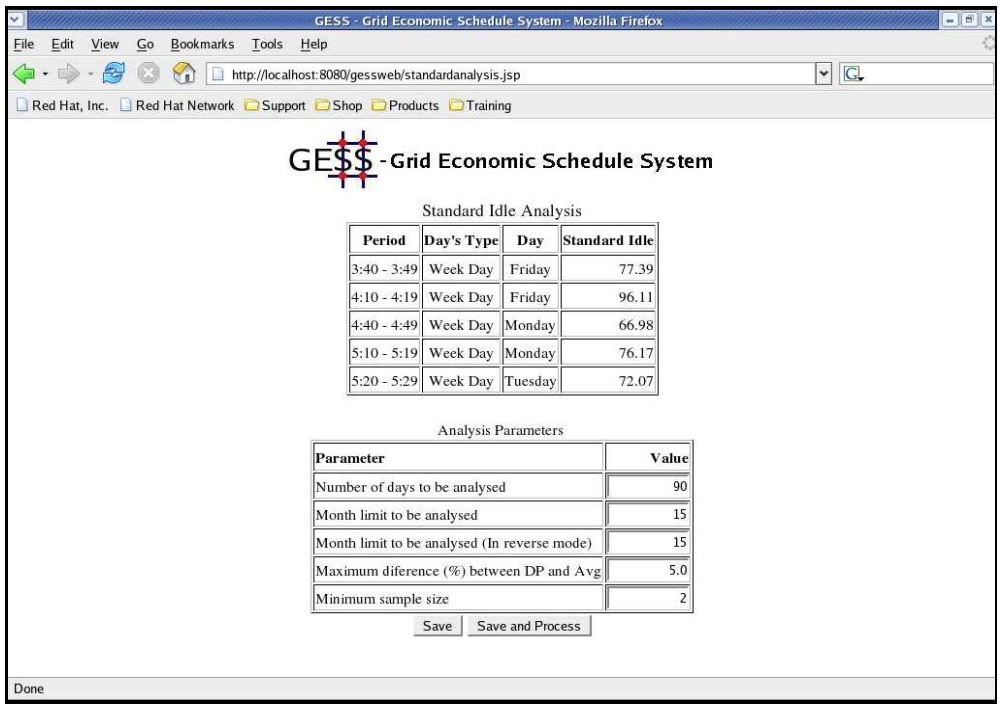


Figura 6.4: Interface de invocação e parametrização do mecanismo de análise de padrões.

Monitoração de Aplicações em Execução

As informações sobre as aplicações de consumidores que estão sendo atualmente executadas em um provedor podem ser obtidas através de uma interface do *PAT* que se comunica com o LLM local. A figura 6.5 apresenta essa interface onde são informados o número do processo criado no sistema operacional para abrigar a aplicação, o nome da aplicação e o percentual de tempo de UCP consumidos.

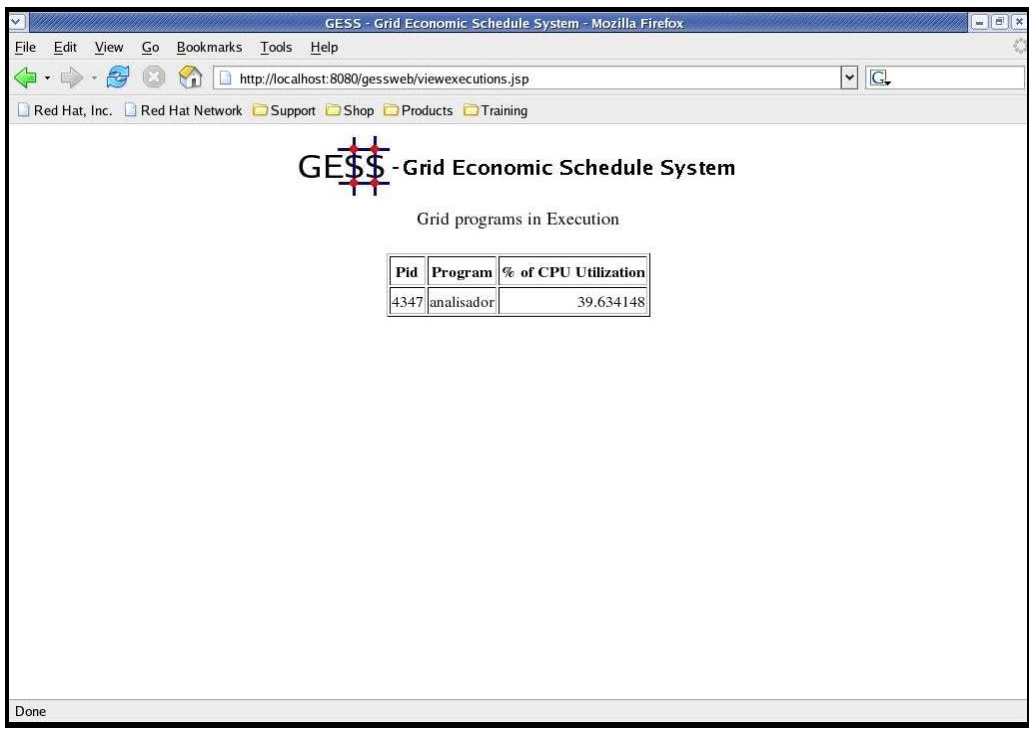


Figura 6.5: Interface para monitoração das aplicações de consumidores em execução.

6.2.7 *Broker*

O componente *Broker* é executado no consumidor e tem a função de localizar os participantes que fornecem um determinado recurso e negociar com eles a execução de uma aplicação.

Para tal, o usuário consumidor deve informar ao Broker um arquivo XML contendo os dados da execução desejada como, por exemplo, aplicações a serem executadas, o tempo

de UCP de cada uma delas, o prazo de execução, o tipo de recurso a ser utilizado, o tipo de otimização e o valor máximo (em G$) a ser gasto. Com base nesse arquivo o *Broker* inicia o processo de negociação.

Após obter a lista dos potenciais provedores, o *Broker* estabelece contato, através de invocações a serviços *Web*, com cada um deles de maneira a negociar pelo recurso. De maneira a minimizar o tempo de negociação, para cada provedor é criada uma *thread* no *Broker* que é responsável por essa negociação.

Com o objetivo de auxiliar o usuário consumidor final, é disponibilizada uma interface GUI (apresentada pela figura 6.6) que, através dos dados informados pelo usuário, gera o arquivo XML e inicia o Broker.

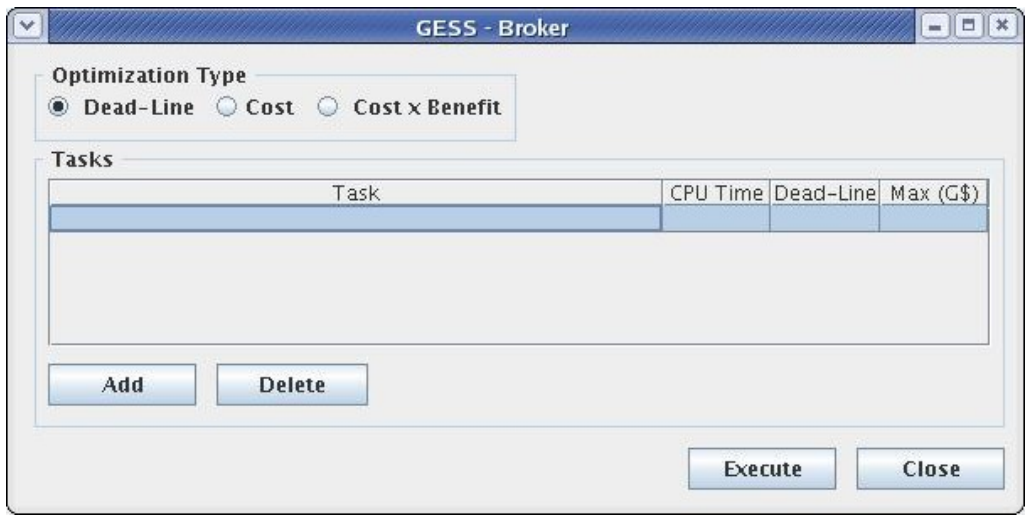


Figura 6.6: Interface para modelagem da execução de aplicações.

Capítulo 7

Conclusões

Com a crescente demanda por maior poder de processamento, a computação em grade tem um potencial de crescimento bastante grande.

No entanto, para promover o crescimento desse paradigma é necessário enfrentar os problemas inerentes à arquitetura distribuída. A existência de participantes independentes, geograficamente distribuídos e sob administrações diferentes propicia um grande número de questões que devem ser tratadas. A arquitetura apresentada nesse projeto trata das questões de qualidade de serviço (QoS) e controle do comportamento egoísta dos participantes.

A inserção de modelos econômicos no tratamento do comportamento egoísta dos participantes incentiva a oferta de recursos, colaborando de forma significativa com o poder computacional da grade. No entanto, cada participante tem a liberdade e flexibilidade para administrar seus recursos da maneira que melhor lhe convir.

O problema existente no fato de um único participante, muitas vezes, ser incapaz de oferecer recursos capazes de gerar a quantidade de créditos necessária para a execução de uma aplicação que consuma um valor de créditos bastante grande foi solucionado através da inserção do conceito de grupos de trabalho. Dessa forma, uma determinada instituição pode organizar seu grupo de trabalho de maneira a atender suas necessidades e garantir que os recursos remotos necessários possam ser utilizados.

A arquitetura proposta foi baseada no *Globus Toolkit* e parcialmente implementada considerando os parâmetros prazo de execução e preço a ser pago por um recurso. As contribuições principais desse trabalho são:

- Mecanismos de regulação automática de preços baseados na oferta, demanda e ta-

xa de utilização de recursos.

- Mecanismos para reserva antecipada de recursos.
- Mecanismos para gerenciamento de grupos de trabalho em que participantes podem ter diversos papéis: coordenador, gerador de créditos e consumidor de recursos. A formação de grupos visa auxiliar consumidores que têm poucos recursos.

Como trabalho futuro, pretende-se desenvolver adaptadores que possibilitem a integração dessa arquitetura com o maior número possível de *middlewares* já existentes para construção de grades computacionais. Em função da preocupação com trabalhos futuros de outros pesquisadores, foram fortemente consideradas as questões de interoperabilidade da arquitetura. Dessa forma, futuras implementações de novos componentes ou melhorias dos já existentes poderão ser efetuadas, desde que utilizem serviços *Web* como forma de integração, mantendo a compatibilidade com o que já está atualmente desenvolvido.

Outros trabalhos futuros incluem:

- Utilização de modelo de componentes no projeto da arquitetura para permitir a reconfiguração conforme os modelos econômicos de interesse dos participantes.
- Avaliação da arquitetura quanto ao seu desempenho.

Além disso, outros atributos de qualidade de serviço podem ser considerados.

Referências Bibliográficas

[1] Buyya, R., Heinz, S., Giddy, J., Abramson, D.: "Economic Models for Management of Resources in Peer-to-Peer and Grid Computing", In Proceedings of SPIE Int. Symposium on The Convergence of Information Technologies and Communications (ITCom 2001), 2001

[2] Eaton, B. C., Eaton, D. F.: "Microeconomia", 3ª Edição, São Paulo, Editora Saraiva, 1999.

[3] Leftwich, R. H.: "O Sistema de Preços e a Alocação de Recursos", 6ª Edição, Editora Bisordi, 1983.

[4] Walras, L.: "Compêndio dos Elementos de Economia Política Pura", 3ª Edição, São Paulo, Editora Nova Cultural, 1988.

[5] Pareto, V.: "Manual de Economia Política", Volume II, São Paulo, Editora Nova Cultural, 1988.

[6] Wellman, M. P.: "A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problems", Journal of Artificial Intelligence Research, 1:1-22, 1993

[7] Werder, M. R.: "Pricing in the Service-Oriented IT World"; Master's Thesis, University of Zurich, 2004.

[8] Fulp, E. W., Ott, M., Reininger, D., Reeves, D. S.: "Paying for QoS: An Optimal Distributed Algorithm for Pricing Networks Resources", In Proceedings of the IEEE Sixth International Workshop on Quality of Service, páginas 75 - 84, 1998.

[9] Vasconcellos, M. A. S,: "Economia: Micro e Macro"; 3ª Edição, São Paulo, Editora Atlas, 2002.

[10] Foster, I., Kesselman, C., Nick, J. M., Tuecke, S.: "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration". Global Grid Forum, 2002.

[11] Foster, I., Kesselman, C., Tuecke, S.: "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal of Supercomputer Applications, Vol. 15, páginas 200-222, 2001.

[12] Buyya, R., Abramson, D., Giddy, J.: "Economic Models for Resource Management and Scheduling in Grid Computing". Concurrency and Computation: Practice and Experience, vol. 14, páginas 1507-1542, 2002.

[13] "Projeto SETI@home". Disponível em: http://setiathome.berkeley.edu. Acessado em Dezembro/2005.

[14] Golle, P., Mironov, I.: "Uncheatable Distributed Computations". Lecture Notes in Computer Science, vol. 2020, page 83, 2001.

[15] Foster, I: "The Grid: A New Infrastructure for 21st Century Science". Physics Today, vol. 55, páginas 42-47, 2002.

[16] Buyya, R., Abramson, D., Venugopal, S.: "The Grid Economy". Proceedings of the IEEE, vol. 93, páginas 698-714, 2005.

[17] Buyya, R., Abramson, D., Giddy, J.: "Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid". HPC Asia, 2000.

[18] Buyya, R., Abramson, D., Giddy, J.: "An Economy Grid Architecture for Service-Oriented Grid Computing", 10th IEEE International Heterogeneous Computing Workshop (HCW 2001), 2001.

[19] Neuman, B. C., Medvinsky, G.: "Requirements for Network Payment: The NetCheque Perspective", In Proceedings of IEEE COMPCON'95, páginas 32-36, 1995.

[20] Buyya, R., Abramson, D., Giddy, J.: "A case for Economy Grid Architecture for Service Oriented Grid Computing". Proceedings 10th IEEE International Heterogeneous Computing Workshop (HCW 2001), San Francisco, California, 2001.

[21] Buyya, R.: "Economic-based Distributed Resource Management and Scheduling for Grid Computing", Ph.D. Thesis, Monash University,2002.

[22] Foster, I.: "Globus Toolkit Version 4: Software for Service-Oriented Systems"; IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, páginas 2-13, 2005.

[23] "About the Globus Toolkit"; Disponível em http://www.globus.org/toolkit/about.html (Acesso em 05/04/2006).

[24] Foster, I., Kesselman, C.: "The Globus Project: A Status Report"; Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop, páginas. 4-18, 1998.

[25] Foster, I., Kesselman, C.: "Globus: A Metacomputing Infrastructure Toolkit"; Intl J. Supercomputer Applications, Vol. 11, páginas 115-128, 1997.

[26] Foster, I.: "A Globus Primer. Or, Everything You Wanted to Know about Globus, but Were Afraid To Ask. Describing Globus Toolkit 4". Disponível em: www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf. Acessado em Julho/2005.

[27] Sotomayor, B.: "The Globus Toolkit 4 Programmer's Tutorial". Disponível em: http://gdp.globus.org/gt4-tutorial/multiplehtml/index.html. Acessado em: Janeiro/2006.

[28] "GT 4.0 WS GRAM Aproach". Disponível em: http://www.globus.org/toolkit/docs/4.0/execution/key/WS_GRAM_Approach.html#id2518665. Acessado em fevereiro/2006.

[29] Madduri, R. K., Hood, C. S., Allcock W. E.: "Reliable File Transfer in Grid Environments". 27th Annual IEEE Conference on Local Computers Networks, 2002.

[30] "Data Management: Key Concepts". Disponível em http://www.globus.org/toolkit/docs/4.0/data/key/index.html#id2488793. Acessado em Fevereiro/2006.

[31] "GridFtp Update January 2002". Disponível em http://www-fp.mcs.anl.gov/dsl/scidac/datagrid/GridFTP%20Overview.pdf. Acessado em Fevereiro/2006.

[32] "Data Management: Key concepts of RLS". Disponível em http://www.globus.org/toolkit/docs/4.0/data/key/rls.html. Acessado em Fevereiro/2006.

[33] "The OGSA-DAI Project". Disponível em: http://www.ogsadai.org.uk/. Acessado em Fevereiro/2006.

[34] "Information Services: Key Concepts". Disponível em: http://www.globus.org/toolkit/docs/4.0/info/key-index.html#id2770079. Acessado em Fevereiro/2006.

[35] "GT 4 Release Notes: Index Service". Disponível em: http://www.globus.org/toolkit/docs/4.0/info/index/WS_MDS_Index_Release_Notes.html#overview. Acessado em Fevereiro/2006.

[36] "GT 4 Release Notes: Trigger Service". Disponível em: http://www.globus.org/toolkit/docs/4.0/info/trigger/WS_MDS_Trigger_Release_Notes.html#s-trigger-Release_Notes-overview. Acessado em Março/2006.

[37] "Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective". Disponível em: www.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf. Acessado em: Março/2006.

[38] "GT 4 Security: Key Concepts". Disponível em: http://www.globus.org/toolkit/docs/4.0/security/key-index.html#s-security-key-overview. Acessado em: Março/2006.

[39] "GT 4: Common Runtime Components". Disponível em: http://www.globus.org/toolkit/docs/4.0/common/. Acessado em: Março/2006.

[40] Coulouris, G., Dollimore, J., Kindberg, T: "Distributed Systems: Concepts and Design". Editora Addison-Wesley, 2000.

[41] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhim, N., Weerawarana, S.: "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI", IEEE Internet Computing, vol. 06, no. 2, páginas. 86-93, 2002.

[42] "SOAP Tutorial". Disponível em: http://www.w3schools.com/soap/default.asp. Acessado em Abril/2004.

[43] Papazoglou, G.: "Service-oriented architectures: approaches, technical and research issues". VLDB Journal, 2006.

[44] Alonso, G., Casati, F., Kuno, H., Machiraju, V.: "Web Services: Concepts, Architectures and Applications", Springer, 2004.

[45] Siegel, J.: "Corba 3: Fundamentals and Programing", Second Edition, Wiley, 2000.

[46] Goldfarb, C. F., Prescod, P.: "The XML Handbook", Second Edition, Prentice-Hall, 2000.

[47] Dantas, M: "Computação Distribuída de Alto Desempenho: Redes, Clusters e Grids Computacionais", Editora Axcel Books do Brasil, 2005.

[48] "Towards Open Grid Services Architecture". Disponível em: http://www.globus.org/ogsa. Acessado em Junho/2006.

[49] "OGSA, OGSI and GT3". Disponível em: http://gdp.globus.org/gt3-tutorial/multiplehtml/ch01s01.html. Acessado em junho de 2006.

[50] "The WS-Resource Framework". Disponível em: www.globus.org/wsrf. Acessado em: junho/2006.

[51] Cirne, W., Neto, E.S.: "Grids Computacionais: da Computação de Alto Desempenho a Serviços sob Demanda", Minicurso, SRBC, 2005.

[52] Foster, I., Czajkowski, K., Ferguson, D. F., Frey, D., Graham, S., Sdukhin, I., Snelling, D., Tuecke, S., Vambenepe, W.: "The WS-Resource Framework". Disponível em: http://www.globus.org/wsrf/specs/ws-wsrf.pdf. Acessado em: Junho/2006.

[53] Brunelle, J., Hurst, P., Huth, J., Kang, l., Ng, C., Parkes, D. C., Seltzer, M., Shank, J., Youssef, S.: "Egg: An Extensible and Economics-Inspired Open Grid Computing Platform". GECON, 2006.

[54] Regev, O., Nisan, N.: "The POPCORN Market – an OnLine Market for Computational Resources". Proceedings of the first international conference on Information and computation economies, páginas 148-157, 1998.

[55] Neto, O. M.: "Entendendo e Dominando o Java". Digeratti Books, 2004.

[56] Bond, M., Haywood, D., Law, D., Longshaw, A., Roxburgh, P.: “Aprenda J2EE”. Makron Books, 2003.

[57] “NetBeans.org”. Disponível em: www.netbeans.org. Acessado em Novembro/2006.

[58] “Web Services – Axis”. Disponível em: ws.apache.org/axis. Acessado em Novembro/2006.

[59] “Apache Tomcat”. Disponível em: tomcat.apache.org. Acessado em Novembro/2006.

[60] “Fedora Project”. Disponível em: fedora.redhat.com. Acessado em Novembro/2006.

[61] Kernigham, B. W., Ritchie, D. M.: “C: A Linguagem de Programação Padrão Ansi”. Editora Campus, 1989.

[62] Tanembaum, A. S.: “Redes de Computadores”. Elsevier, 2003.

[63] Horstmann, C. S., Cornell, G.: “Core Java 2: Fundamentos”, Makron Books, 2001.

[64] Horton, I.: “Begining Java”, Wrox, 1997.

[65] Tanembaum, A. S., Woodhull, A. S.: “Sistemas Operacionais: Projeto e Implementação”, Segunda Edição, Bookman, 2000.

[66] Card, R., Dumas, E., Mével, F.: “The Linux Kernel Book”, Wiley, 1998.

[67] “PostgreSQL BR”, Disponível em: http://www.postgresql.org.br. Acessado em novembro/2006.

[68] OURGRID Home-Page. Disponível em: www.ourgrid.org. Acessado em Dezembro/2006.

[69] Andrade, N., Costa, L., Germóglio, G., Cirne, W.: “Peer-to-Peer grid computing with the OurGrid Community”, SBRC, Fortaleza-CE, 2005.

[70] Cirne, W., Brasileiro, F., Sauvé, J., Andrade, N., Paranhos, D., Santos-Neto, E., Medeiros, R.: “Grid Computing for Bag of Tasks”, Third IFIP Conference on E-Commerce, E-Business and E-Goverment, São Paulo-SP, páginas 591-609, 2003.

[71] Costa, L.B., Feitosa, L., Araújo, E., Mendes, G., Coelho, R., Cirne, W., Fireman, D.: “MyGrid – A complete solution for running Bag-of-Tasks Applications”. SBRC, 2004.

[72] Santos, R., Andrade, A., Cirne, W., Brasileiro, F., Andrane, N.: “Accurate Autonomous Accounting in Peer-to-Peer Grids”, 3rd International Workshop on Middleware for Grid Computing, Grenoble (France), páginas 1-6, 2005.

Apêndice A

Demonstração da Variação do Preço

Esse apêndice ilustra, através de tabelas, o comportamento do preço base de um recurso em alguns possíveis cenários.

A.1 Oferta Fixa e Demanda em Crescimento

Nessa seção considera-se um cenário onde os comportamentos da demanda e da oferta são mensurados para ajuste do preço do recurso, sendo somente a demanda alterada. Foi selecionado como velocidade do ajuste o valor 0,5 tanto para demanda quanto para oferta (variáveis a e β da fórmula ilustrada na figura 5.3).

Tabela A.1: Demonstração do Comportamento do Preço de um Recurso

Período	Demanda	Oferta	Utilização (%)	Preço (G$)
1	100	100	50	10
2	110	100	50	10,50
3	120	100	50	10,98
4	130	100	50	11,44
5	140	100	50	11,88
6	150	100	50	12,30
7	160	100	50	12,71
8	170	100	50	13,11

A.2 Demanda Fixa e Oferta em Crescimento

Nessa seção considera-se um cenário onde os comportamentos da demanda e da oferta são mensurados para ajuste do preço do recurso, sendo somente a oferta alterada. Foi selecionado como velocidade do ajuste o valor 0,5 tanto para demanda quanto para oferta (variáveis *a* e *β* da fórmula ilustrada na figura 5.3).

Tabela A.2: Demonstração do Comportamento do Preço de um Recurso

Período	Demanda	Oferta	Utilização (%)	Preço (G$)
1	100	100	50	10
2	100	110	50	9,50
3	100	120	50	9,07
4	100	130	50	8,69
5	100	140	50	8,36
6	100	150	50	8,06
7	100	160	50	7,79
8	100	170	50	7,55

A.3 Relação entre Oferta e Demanda

Nessa seção considera-se um cenário onde a relação entre a demanda e a oferta é mensurada para ajuste do preço do recurso. A partir do 5º período encontra-se uma situação de equilíbrio, onde o preço fica inalterado. Foi selecionado como velocidade do ajuste o valor 0,5 (variáveis *a* e *β* da fórmula ilustrada na figura 5.3).

Tabela A.3: Demonstração do Comportamento do Preço de um Recurso

Período	Demanda	Oferta	Utilização (%)	Preço (G$)
1	100	140	50	10
2	100	130	50	10,36
3	100	120	50	10,76
4	100	110	50	11,20
5	100	100	50	11,71
6	90	90	50	11,71
7	80	80	50	11,71
8	70	70	50	11,71

A.4 Recurso Sub-Utilizado

Nessa seção considera-se um cenário onde a oferta e demanda se mantém. No entanto, a taxa de utilização do recurso, a qual é considerada para o ajuste do preço, é baixa (fato que sinaliza que seu preço pode estar acima dos praticados pelos demais provedores). Foi selecionado como velocidade do ajuste o valor 0.5 (variável *d* da fórmula ilustrada na figura 5.3).

Tabela A.4: Demonstração do Comportamento do Preço de um Recurso

Período	Demanda	Oferta	Utilização (%)	Preço (G$)
1	100	100	50	10
2	100	100	55	7,75
3	100	100	60	6,20
4	100	100	65	5,12
5	100	100	70	4,35
6	90	100	75	3,80
7	80	100	80	3,42

8	70	100	85	3,17

A.5 Recurso Super-Utilizado

Nessa seção considera-se um cenário onde a oferta e demanda se mantém. No entanto, a taxa de utilização do recurso, a qual é considerada para o ajuste do preço, é total no primeiro período (fato que sinaliza que seu preço pode estar abaixo dos praticados pelos demais provedores). Sendo assim, a partir desse período o preço é ajustado, influenciando as taxas de utilização dos demais períodos. Foi selecionado como velocidade do ajuste o valor 0,5 (variável γ da fórmula ilustrada na figura 5.3).

Tabela A.5: Demonstração do Comportamento do Preço de um Recurso

Período	**Demanda**	**Oferta**	**Utilização (%)**	**Preço (G$)**
1	100	100	100	10
2	100	100	90	15
3	100	100	92	14,25
4	100	100	94	13,68
5	100	100	96	13,27
6	90	100	98	13,00
7	80	100	99	12,87
8	70	100	99	12,81