Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Programa de Pós-Graduação em Ciência da Computação

Um Escalonador Orientado a Sites para Grades Computacionais

Rodrigo Neves Calheiros

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação

Orientador: Prof. Dr. César Augusto Fonticielha De Rose

Porto Alegre
2006

Dados Internacionais de Catalogação na Publicação (CIP)

C152e Calheiros, Rodrigo Neves
Um escalonador orientado a sites para grades computacionais / Rodrigo Neves Calheiros. – Porto Alegre, 2006.
83 f. : il.

Dissertação (Mestrado) – Fac. de Informática, PUCRS, 2006.

1. Computação em Grade. 2. Arquitetura de Computadores. 3. Escalonamento – Processos. 4. Web Sites – Programação. 5. Sistemas Distribuídos. 6. Gerência de Recursos. I. Título.

CDD 004.22

Ficha Catalográfica elaborada pelo
Setor de Processamento Técnico da BC-PUCRS

Aos meus pais, Alcides e Maria.

Agradecimentos

Ao meu orientador, prof. César De Rose, por me dar a chance de tornar isto possível, e por todos os conselhos, dicas e oportunidades; ao prof. Tiago Ferreto, cujos conselhos e apoio teórico foram fundamentais para o desenvolvimento deste trabalho; a todos os amigos e colegas do CPAD, com os quais aprendi muito e compartilhei um ótimo ambiente de trabalho ao longo destes quase dois anos; à minha família, que sempre me apoiou de todas as formas possíveis; a todos os meus amigos, cuja presença (real ou virtual) propiciou um alívio para toda tensão desta etapa de minha vida; ao pessoal do LSD/UFCG (desenvolvedores do OurGrid) pelo suporte técnico e pela hospitalidade e cortesia com que fui recebido em Campina Grande; à HP Brasil, por apoiar o projeto. Por último, e não menos importante, um agradecimento especial à Juliana, a verdadeira motivação de todo o meu trabalho.

Resumo

Este trabalho apresenta uma nova abordagem para escalonamento de recursos em grades computacionais. O Site Resource Scheduler (SRS) tem por objetivo simplificar a visão que usuários têm dos recursos da grade e a sua gerência. Decisões de escalonamento de recursos são movidas para o site que os hospeda, permitindo reposta rápida a mudanças. Usuários não conhecem os recursos que utilizam, mas enxergam uma "máquina virtual" que representa a quantidade de processamento a eles disponível naquele site. Esta abordagem inclui novos desafios, como escalonamento em dois níveis e a necessidade de fornecer uma medida de capacidade computacional. Porém, ele simplifica e otimiza o escalonamento em grades. Neste trabalho será definida a arquitetura do SRS, apresentado um protótipo desenvolvido para o middleware de grade OurGrid, os respectivos testes de desempenho e descrita uma forma de implantar o SRS em grades Globus.

Palavras-chave: Computação em Grade. Gerência de Recursos. Sistemas Distribuídos.

Abstract

This work presents a novel approach to resource scheduling in computational grids. The Site Resource Scheduler (SRS) aims at simplifying the vision users have from grid resources and their management. Scheduling decisions are moved to the site where resources are hosted, allowing quick response to changes. Users do not need to be aware of the resources they use and are instead supplied with a "virtual machine" that represents the amount of processing power available to them in the site. This approach results in new challenges, like management of two level scheduling scheme and the need to define a computational power unit. However it simplifies and optimizes scheduling in Grids. In this work the SRS architecture is defined, a prototype for the OurGrid middleware with some performance results is presented and a method to deploy SRS in a Globus grid is described.

Keywords: Grid Computing. Resource Management. Distributed Systems.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

BoT	*Bag of Tasks*	24
COW	*Cluster of Workstations*	21
CRM	*Cluster Resource Manager*	54
CSF	*Community Scheduler Framework*	32
EPR	*End Point Reference*	33
FTP	*File Transfer Protocol*	32
GGF	*Global Grid Forum*	29
GRAM	*Grid Resource Allocation and Management*	35
LDAP	*Lightweight Directory Access Protocol*	30
MDS	*Monitoring & Discovery System*	32
MPI	*Message Passing Interface*	24
OGSA-DAI	*Open Grid Services Architecture Data Access and Integration*	32
OGSA	*Open Grid Services Architecture*	35
OGSI	*Open Grid Services Infrastructure*	30
RFT	*Reliable File Transfer*	32
RSL	*Resource Specification Language*	69
SLA	*Service Level Agreement*	24
SOAP	*Simple Object Access Protocol*	31
SRS	*Site Resource Scheduler*	21
TLS	*Transport Layer Security*	31
VO	*Virtual Organization*	25
WQR	*WorkQueue with Replication*	35
WSDL	*Web Services Definition Language*	31
XML	*eXtensible Markup Language*	31

Sumário

1 Introdução

A computação em grade é uma área de pesquisa da ciência da computação que deixou de ser uma simples promessa acadêmica para se tornar uma importante ferramenta para a ciência e para a indústria. Cada vez mais, grandes empresas apóiam projetos relacionados à computação em grade, ao mesmo tempo em que cada vez mais pesquisadores de áreas como física, medicina, engenharia, química e biologia adotam a grade como ferramenta para seus estudos. Apesar de todo o sucesso na adoção desta tecnologia, muita pesquisa ainda é necessária para que possam ser aproveitadas todas as possibilidades oferecidas pelas grades.

Uma destas áreas onde pode haver evolução no estado da arte é a área de gerência e escalonamento de recursos em grades. Considera-se o gerente de recursos o agente da grade responsável por localizar, negociar e monitorar recursos [1]. Em outras palavras, é o gerente de recursos que disponibiliza à grade a capacidade computacional que a torna tão atraente a cientistas e pessoas de negócios.

O escalonamento de recursos em grades, o processo de submeter aplicações a recursos sobre os quais o usuário não tem controle [2] pode ser realizado de duas formas: a primeira, orientada ao usuário, tem como objetivo otimizar a execução da aplicação de um usuário, através da redução do seu tempo de execução. Isto é conseguido com a escolha dos melhores recursos disponíveis para o usuário.

Obviamente, há um conflito entre escalonadores de usuários, pois cada um tentará alocar para si os melhores recursos disponíveis, gerando uma disputa por recursos na grade. Há uma segunda forma de escalonamento em grades, orientada ao sistema, onde o objetivo é otimizar a utilização dos recursos, sem procurar beneficiar nenhum usuário em especial. Em geral, estes últimos escalonadores (conhecidos como *metaschedulers*) são utilizados entre organizações virtuais, permitindo que participantes destas organizações compartilhem seus recursos da forma mais igualitária possível, dentro de condições pré-estabelecias pela comunidade.

Neste trabalho é apresentado o *Site Resource Scheduler* (SRS), um escalonador e gerente de recursos que objetiva simplificar a visão que usuários têm dos recursos da grade através da sua virtualização. Considera-se que um *site* possui recursos, que podem ser utilizados por *usuários locais*, isto é, usuários que acessam diretamente os recursos e *usuários da grade*, que acessam os recursos a partir de um *software* de grade. Os recursos do *site* podem ser dedicados, quando são utilizados exclusivamente por um usuário por um tempo determinado ou não-dedicados, quando podem ser compartilhados entre usuários. Um exemplo típico de recursos dedicados são *clusters of workstations* (COW) [3] e de recursos não-dedicados são estações de trabalho.

A otimização da utilização dos recursos é obtida através da virtualização dos recursos, es-

condendo dos usuários da grade a complexidade da gerência de recursos que podem ser de diferentes arquiteturas, com diferentes modos de acesso e que não são dedicados.

1.1 Grades computacionais

O termo “grade”, aplicado no contexto de grades computacionais, surgiu em meados da década de 90 para descrever uma infra-estrutura de computação distribuída voltada a aplicações científicas [4]. Desde então, diversas definições para o termo surgiram na literatura.

Foster et al. [4] definem grade como um ambiente capaz de promover compartilhamento de recursos de forma segura, coordenada e flexível entre indivíduos, instituições e recursos. Segundo Berman et al. [5], grade é uma infra-estrutura computacional e de gerenciamento de dados que integra diversas tecnologias para fornecer uma plataforma virtual de computação. Chetty e Buyya [6] definem grades como “*Redes baseadas em Internet de recursos computacionais geograficamente distribuídos que cientistas podem compartilhar, selecionar e agregar para resolver problemas de larga escala*”. Recentemente grandes empresas de tecnologia também começaram a manifestar interesse pela tecnologia, o que permitiu que aplicações científicas deixassem de ser aplicações predominantes nestes ambientes, abrindo caminho para aplicações voltadas a negócios [7].

Em comum entre todas as definições de grades citadas, está a necessidade do compartilhamento de recursos distribuídos entre diferentes instituições, que possuem diferentes políticas de uso e de segurança nos seus recursos. Essa heterogeneidade de recursos e políticas associado ao fato de que estas organizações podem estar geograficamente distantes é que torna a computação em grade uma área desafiadora e ao mesmo tempo muito promissora. Algumas outras características de grades são as seguintes [8–10]:

Escalabilidade. O tamanho de uma grade pode crescer, evoluindo de um sistema com alguns elementos para sistemas contendo milhões de recursos;

Heterogeneidade. Os componentes da grade são heterogêneos, podendo ser formados por computadores de diversas arquiteturas executando sobre diferentes plataformas e possuindo diferentes características;

Compartilhamento. Diferentes aplicações devem ter direito de utilização da grade, não podendo ela ser destinada a uma única aplicação;

Controle distribuído. Não existe uma entidade central com controle sobre toda a grade;

Repositório virtual de recursos. Os recursos (não somente processadores) são vistos pelos usuários como dispostos em um repositório virtual de onde eles podem alocar recursos. O acesso não é feito ao recurso diretamente, e sim ao repositório que fornece o recurso;

Dinamicidade. A qualquer momento algum recurso pode se tornar indisponível, em virtude de falhas que podem ocorrer em componentes da grade. O sistema tem de ser capaz de lidar com estas falhas de forma eficiente;

Transparência. A grade não deve interferir na autonomia ou administração do local onde está instalada; não deve comprometer a sua segurança nem exigir substituição de *software* previamente existente no local. Do ponto de vista do usuário da grade, deve-se permitir o seu acesso e partida do ambiente conforme a sua vontade e não deve lhe impor linguagens, ferramentas, bibliotecas e paradigmas de programação.

Projetos de sistemas de grades computacionais devem levar em conta alguns aspectos, como os seguintes:

Segurança. Por sua característica de distribuição entre diversos domínios administrativos, é necessário garantir que não ocorrerá um acesso não autorizado aos recursos da grade [8]. Também é esperado que o acesso de usuários da grade aos recursos de um *site* seja facilitado, não exigindo todo o procedimento convencional de cadastro que ocorrem em *sites* convencionais. Tudo isso, porém, deve ser feito respeitando-se os sistemas de segurança previamente existentes no local. De forma similar, a adição de novos mecanismos de segurança na grade deve respeitar os critérios de segurança locais, e fazê-lo sem exigir a intervenção do administrador do sistema [11].

Economia. Um aspecto importante a ser considerado dentro de um ambiente de grade é a de compartilhamento de recursos. Deve existir uma motivação para que uma organização disponha seus recursos à grade. A economia de grade discute mecanismos de compra e venda virtual de recursos, de forma que, na medida do possível, satisfazendo tanto "compradores" quanto "vendedores" [12]. Em um modelo econômico mais simples, pode ser formada uma rede de "troca de favores", onde uma organização cede recursos à grade esperando, da mesma forma, receber recursos de outras organizações quando deles necessitar. Modelos mais complexos podem envolver mecanismos de compra e venda baseados em modelos de negócios reais, como leilões e licitações [12].

Imagem de sistema. Grades devem oferecer uma imagem transparente do sistema, através da abstração da sua heterogeneidade. Comumente, o problema é atacado de duas formas [8]: em nível de usuário, utilizando-se as abstrações que já são conhecidas pelo usuário e através do desenvolvimento de novas abstrações. A vantagem da primeira abordagem é a de não exigir o desenvolvimento de novos sistemas, à custa de uma possível perda de desempenho. A segunda tende a ser mais eficiente, mas exige do usuário um aprendizado, além de ser necessária a incorporação da nova característica no sistema existente.

Tolerância a falhas. Grades computacionais podem abranger um número de recursos da ordem de milhões de elementos. Mesmo que a probabilidade individual de falha de cada

um deles seja pequena, a probabilidade de ocorrência de falha em um dos elementos é alta, exigindo que essa possibilidade seja considerada pelos componentes do *middleware* da grade. As falhas podem ocorrer devido a diversos fatores, desde a perda de recursos em favor de usuários locais a erros de configuração do sistema [13]. Os sistemas de grade atuais exigem um grande esforço do usuário para detecção do motivo da falha [13], o que indica que esta é uma área que ainda necessita de maiores pesquisas.

Aplicações. A classe de aplicações suportada por um sistema em grade deve ser levada em consideração por desenvolvedores de aplicações nestas plataformas. Entre as aplicações fracamente acopladas (amplamente suportadas por ambientes de grade) encontram-se aplicações *parameter sweep* [14], compostas por diversas instâncias de um mesmo programa, executando com diferentes parâmetros, aplicações BoT (*Bag of Tasks* ou Saco de Trabalho), cujas tarefas que a compõem são independentes umas das outras [15] e aplicações *workflow* [16], onde existe uma relação de ordem parcial na execução das tarefas que compõem a aplicação, sendo a ordem determinada por alguma dependência de dados ou de controle entre as tarefas. Uma grade pode suportar também a execução de aplicações fortemente acopladas, onde as tarefas podem se comunicar durante a execução. Um exemplo deste tipo de aplicação são as aplicações que utilizam a biblioteca MPI (*Message Passing Interface*) [17] de troca de mensagens.

Gerência de recursos. A forma como recursos e serviços fornecidos por uma grade são disponibilizados para outras entidades é a questão trabalhada em gerência de recursos [1]. Em grades, esta questão é mais complexa porque envolve recursos heterogêneos e compartilhamento entre diferentes domínios administrativos, que podem possuir diferentes políticas para o assunto. Outras questões, como a possibilidade de co-escalonamento (alocação simultânea de diferentes recursos), *advanced reservation* (possibilidade de realizar reservas de recursos para uso em algum momento no futuro) e SLA (*Service Level Agreement*) – a capacidade de negociar qualidade de serviço para recursos da grade [18] – também têm sido investigadas por grupos de pesquisa relacionados à grades.

Escalonamento de recursos. Escalonadores para grades computacionais devem ser capazes de escalonar aplicações de usuários em recursos geograficamente distribuídos, sobre os quais não possuem controle direto. Escalonadores de grades podem ser *orientados à aplicação* (quando realizam alocações com base nas necessidades do usuário, sem levar em consideração o sistema como um todo) [19] ou *orientados ao sistema* (quando o escalonamento objetiva fornecer uma utilização balanceada do sistema entre os usuários). *Grid scheduling* é o processo de escalonar recursos distribuídos em múltiplos domínios administrativos [2], realizando as tarefas de descoberta (localização de recursos que satisfaçam os requisitos da aplicação) e de mapeamento (determinação de quais recursos executarão quais aplicações) de recursos, de forma orientada à aplicação. Este

processo é realizado por cada usuário da grade que deseja obter recursos. Um elemento capaz de realizar as mesmas atividades de forma orientada ao sistema é conhecido como *metascheduler* [20], que pode ser organizado de três formas [21, 22]: *Centralizada* (quando o *metascheduler* submete aplicações diretamente para o escalonador local, que recebe a tarefa e submete aos recursos especificados), *hierárquica*, (quando o *metascheduler* encaminha aplicações aos escalonadores locais de cada domínio, e cabe a estes decidir sobre o seu escalonamento local) e *distribuída* (quando em cada domínio existe um *metascheduler* local que pode executar aplicações ou enviá-las para outros domínios, de acordo com critérios de carga e de desempenho). O *metascheduler* recebe pedidos de recursos de *grid schedulers* e realiza o escalonamento dos recursos. A Figura 1 apresenta cada um dos três esquemas apresentados.

Heurísticas de escalonamento. A forma como são escolhidos os processadores que executarão as tarefas que compõem uma aplicação é o aspecto mais problemático do escalonamento em grades, pois é sabido que a tarefa de escalonar processos em processadores heterogêneos é um problema NP-Completo [23]. Esta constatação obriga os pesquisadores da área de escalonamento em grades a procurar soluções heurísticas para o problema. A comparação destas heurísticas é complicada pelo fato de que cada pesquisador que apresenta uma nova heurística realiza testes com considerações diferentes daquelas assumidas por algum outro pesquisador. Ainda com relação a estas heurísticas, nem sempre é possível aplicá-las em todos os tipos de grades: algumas consideram o modelo centralizado de *metascheduling*, outras consideram um modelo hierárquico; algumas são para serem aplicadas de forma *on-line*, outras consideram um escalonamento em lote (*batch*). Uma apresentação detalhada das principais heurísticas para escalonamento de recursos em grades computacionais podem ser obtidas em [14, 22–25].

1.1.1 Organizações Virtuais

Um outro conceito relacionado à computação em grade muito presente na literatura sobre o assunto é o conceito de VO's – *Organizações Virtuais* (do inglês *Virtual Organizations*).

VO's podem ser definidas como um conjunto de indivíduos pertencentes a diversas organizações clássicas (empresas, universidades), possivelmente distribuídos geograficamente, e seus respectivos recursos que serão compartilhados de maneira organizada, por um tempo que pode ser determinado ou não [4, 11].

Como muitos dos cenários propostos para grades se baseiam no conceito de VO's, as Organizações Virtuais voltarão a ser referenciadas durante o texto.

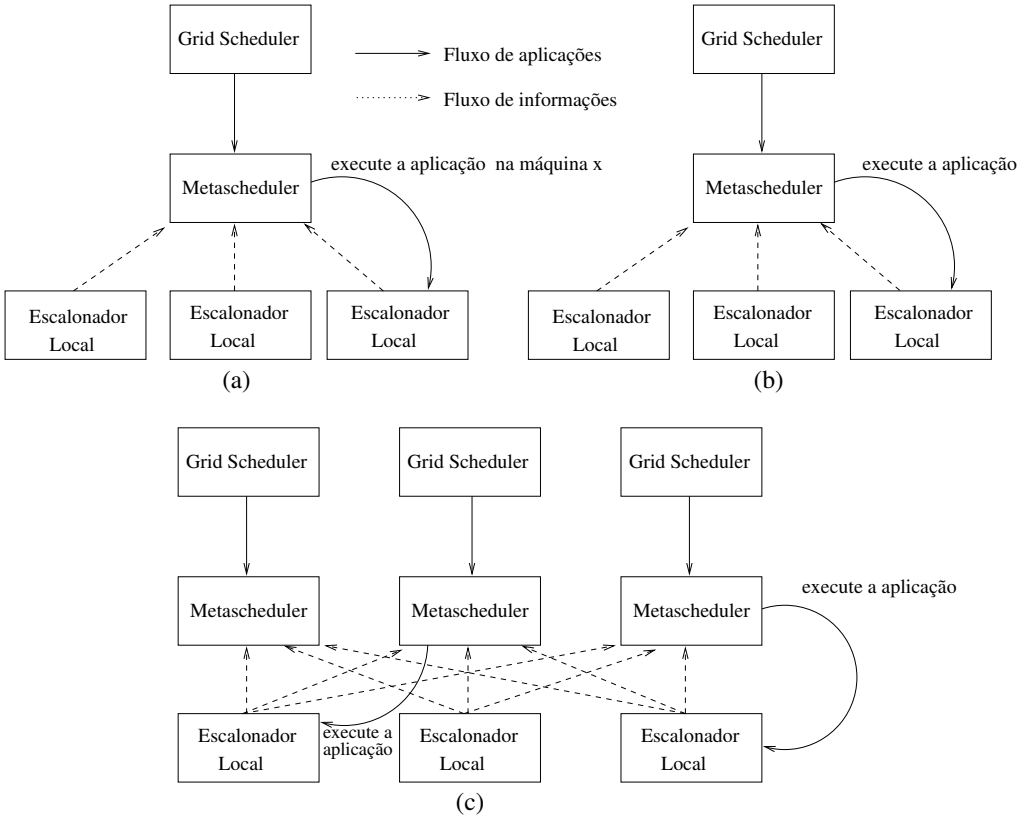

Figura 1 – Organizações de *metaschedulers* (a) Centralizada (b) Hierárquica (c) Distribuída.

1.1.2 Aplicações e tarefas

No contexto deste trabalho, será utilizado o termo “aplicação” para referenciar o trabalho enviado para execução por um usuário. Uma aplicação (*job*) é composta por uma ou mais tarefas (*tasks*). Que podem ser independentes (no caso de aplicações fracamente acopladas) ou podem possuir alguma comunicação entre elas (em aplicações fortemente acopladas).

1.2 Objetivos

A objetivo deste trabalho é de fornecer um meio de simplificar a visão que usuários têm dos recursos da grade e conseqüentemente a sua gerência, transferindo decisões de escalonamento de recursos para o site que os hospeda e com isto reduzindo o custo do escalonamento e gerência destes recursos por parte dos usuários, ao mesmo tempo em que aumenta a escalabilidade de seus *grid schedulers.*

Isto é atingido através da adição de uma camada de abstração sobre o escalonamento na grade, realizado em nível de *site*: os recursos de um *site* são virtualizados e apresentados de forma abstrata aos usuários. Desta forma, o usuário deixa de controlar dezenas (talvez centenas

ou milhares) de recursos e passa a controlar uma menor quantidade de recursos que representam a capacidade computacional de cada um dos *sites* aos quais tem acesso.

A virtualização pode trazer outras vantagens em *sites* onde os recursos são fornecidos à grade de forma oportunística: a eventual indisponibilidade do recurso pode ser tratada localmente e ser mantida transparente ao usuário. Para isto, o *site* deve possuir *softwares* capazes de monitorar o estado dos seus recursos, de re-executar tarefas que não completaram devido a esta indisponibilidade e de armazenar arquivos submetidos pelo usuário, transferindo estes arquivos para os recursos que executam a aplicação.

A arquitetura proposta para a realização destas tarefas – chamada de *Site Resource Scheduler* (SRS) será apresentada. A sua implantação pode ocorrer como um *middleware* independente para grades ou pode ser aplicado aproveitando-se os serviços oferecidos por algum *middleware* existente.

Embora a arquitetura apresentada possa abordar quaisquer dos itens listados na Seção 1.1, a ênfase deste trabalho será na parte de virtualização e gerência dos recursos, pois estes são os aspectos onde o SRS fornece uma contribuição original, abordando o problema de uma forma diferente da convencional. Outros aspectos como segurança, economia e escalonamento serão abordados em termos mais gerais.

1.3 Organização do texto

Este documento está organizado da seguinte forma. No Capítulo 2 é apresentado o estado da arte em grades computacionais. São apresentados os principais sistemas disponíveis e algumas tendências em escalonamento e gerência de recursos em grades. No Capítulo 3 o *Site Resource Scheduler* é apresentado juntamente com a sua arquitetura. No Capítulo 4 será apresentado um protótipo do SRS desenvolvido como estudo de caso para o *middleware* OurGrid, os testes realizados com o protótipo e os resultados obtidos. No Capítulo 5 é discutida uma forma de implantar o SRS em uma grade que utiliza o *middleware* Globus, enquanto a conclusão e alguns possíveis desdobramentos do trabalho são apresentados no Capítulo 6.

2 Estado da arte

Apesar de ser uma área de pesquisa recente em ciência da computação (o termo “grade” dentro do contexto aqui utilizado surgiu em meados da décadas de 90 [4]), a computação em grade é uma área bastante ativa e em evolução. A nomenclatura ainda não é bem definida, surgindo eventualmente diferenças de terminologia entre pesquisadores. O *Global Grid Forum* (GGF) é uma organização que congrega indústria e academia e que procura desenvolver padrões para grades computacionais [26]. Diversas especificações estão sendo propostas pelo GGF com o objetivo de uniformizar a terminologia e a visão de grade, permitindo o desenvolvimento de *software* interoperável.

Neste capítulo serão discutidos aspectos de estado da arte em grades computacionais, especificamente naqueles assuntos que são de maior importância para este trabalho: *middleware* de grades computacionais e gerência e escalonamento de recursos nesta plataforma.

2.1 Middleware para grades computacionais

Diversos projetos de sistemas de software (*middleware*) para grades computacionais têm sido desenvolvidos nos últimos anos. Estes projetos ocorrem em diversas partes do mundo e com diferentes propósitos.

Em geral, os projetistas deste tipo de sistema costumam otimizá-los para algum cenário específico de hardware (isto é, as características físicas e organizacionais dos recursos que comporão a grade) e de software (isto é, o tipo de aplicações suportado pelo ambiente). Alguns exemplos de *middleware* para grades são:

UNICORE. O UNICORE (*Uniform Interface to Computing Resources*) é um *middleware* para grades computacionais que suporta aplicações *workflow* [27]. Como principal resultado deste sistema, tem-se o EUROGRID, uma grade que interliga centros de pesquisas da Europa. O escalonamento no UNICORE é manual: o usuário define os requisitos de cada aplicação e determina o recurso onde deseja que seja executada. O *software* cliente procura mapear tarefas para recursos que atendam às necessidades da aplicação, cabendo ao usuário a indicação de novos recursos se os disponíveis não atenderem aos seus requisitos.

Entropia. Entropia é um sistema proprietário para grades computacionais voltado a ambientes corporativos. Ele é composto pelo Entropia 2000, que explora ciclos ociosos de máquinas

através da Internet e pelo DCGRID 5.0, que explora ciclo ociosos de *desktops* presentes em organizações [28]. Um diferencial do Entropia para as demais plataforma é o fato de ser desenvolvido para ser utilizado em máquinas que utilizam o sistema operacional Windows.

Avaki/Legion. O Legion [29] é um dos projetos pioneiros para implementação de *middleware* para grades. Os projetistas do Legion buscavam características bem definidas durante a sua elaboração: autonomia dos domínios, núcleo extensível, arquitetura escalável, facilidade de uso, alto desempenho via paralelismo, espaço de objetos único e extensível, segurança, gerenciamento e aproveitamento da heterogeneidade, suporte a múltiplas linguagens e interoperabilidade e tolerância a falhas [29]. Para executar neste ambiente, as aplicações precisam ser compiladas com uma biblioteca própria. Este sistema implementa um escalonamento de tarefas aleatório, mas permite que usuários desenvolvam heurísticas elaboradas. Em 2001, o *software* foi renomeado para Avaki e passou a ser disponibilizado comercialmente [30].

Dois *softwares* que possuem uma relevância maior neste trabalho, o Globus e o OurGrid, serão apresentados de forma mais detalhada a seguir.

2.1.1 Globus

Um dos projetos pioneiros em computação em grade, o Globus é um conjunto de ferramentas que provê uma infra-estrutura para grades. As ferramentas do Globus implementam serviços básicos para diversas características da grade, tais como segurança, localização, alocação e gerência de recursos, informação, acesso aos dados e execução remota [31].

Uma característica importante do Globus é a sua modularidade: não é necessário que todos os serviços do Globus sejam utilizados simultaneamente. Serviços podem ser ativados conforme a necessidade do usuário do sistema [32].

As primeiras versões do Globus (Globus 1 e Globus 2) utilizam para comunicação a biblioteca Nexus [32]. A versão 3 do *software* utiliza *web services* [33] para implementar os serviços mais básicos, através do OGSI (*Open Grid Services Infrastructure*), uma implementação da arquitetura OGSA proposta pelo GGF [7]. Ao utilizar o OGSI, evita-se a utilização de protocolos específicos para cada ação a ser realizada sobre a grade, por exemplo LDAP (*Lightweight Directory Access Protocol*) para informações.

A versão atual do Globus, o Globus 4, implementa o *WS-Resource Framework*, uma especificação para *web services* com estado (*stateful*) que substitui o OGSI e é apoiado por empresas como IBM e HP [34]. A Figura 2 (adaptada de [31]) mostra a arquitetura do Globus 4. Nesta figura, as aplicações do lado do cliente são aplicações que utilizam os serviços do Globus, como por exemplo *grid schedulers.* Clientes também podem desenvolver serviços de grade, que exe-

cutarão em um contêiner. A comunicação entre aplicações cliente e servidor é baseada em *web services*, com exceção de três serviços: GridFTP e MyProxy – que utilizam protocolos próprios – e RLS, que utiliza *Transport Layer Security* – TLS [35] – um protocolo para comunicações seguras.

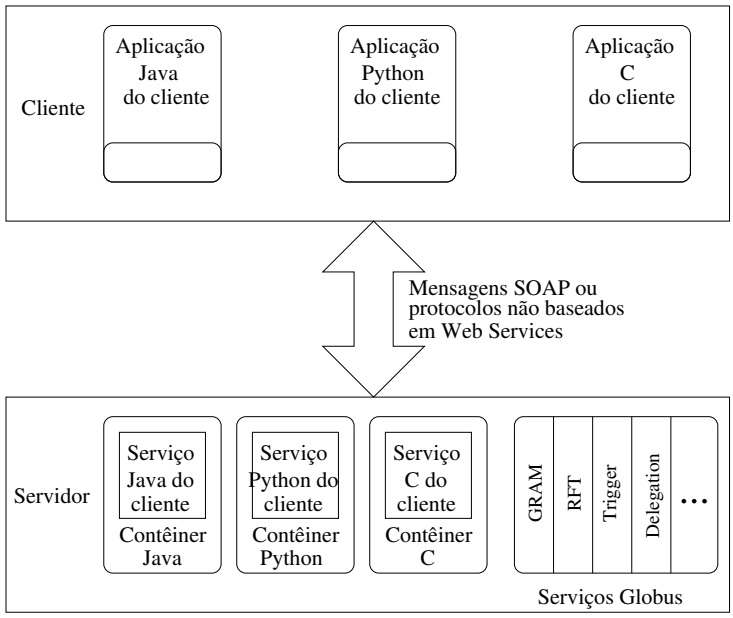

Figura 2 – Arquitetura do Globus Toolkit 4.

A utilização de *web services* para comunicação possibilita que seja utilizado um padrão bem definido para as mensagens trocadas entre as partes (XML – *eXtensible Markup Language* [36]) e para descrição de serviços (WSDL – *Web Services Definition Language* [33]). Contudo, serviços do Globus não baseados em *web services* ainda são suportados pela nova versão do *software*.

Os serviços Globus podem ser divididos em cinco grupos: segurança, gerência de dados, gerência de execução, serviços de informação e componentes de tempo de execução. A seguir são descritos os componentes baseados em *web services* que pertencem a cada grupo [31].

Segurança. Quatro componentes do Globus pertencem a este grupo. O primeiro, responsável por intermediar as transação entre clientes da grade e recursos, permitindo a definição de direitos de acesso a usuários e facilitando o seu acesso aos recursos, é o CAS (*Community Authorization Service*). O *delegation service* é responsável por delegar credenciais de acesso do usuário aos recursos, permitindo que novos serviços utilizem as mesmos credenciais de usuários e permitindo a sua renovação. O terceiro serviço, o *web service authentication & authorization*, permite o envio de mensagens seguras através do SOAP (*Simple Object Access Protocol*), protocolo de troca de mensagens utilizado em *web services* [33]. O último serviço é o de gerência de credenciais (*credential management*).

Gerência de dados. O RFT (*Reliable File Tranfer*) é um serviço para transferência de arquivos seguro baseado em *web services*, que por sua vez utiliza os serviços do GridFTP, um protocolo FTP (*File Transfer Protocol*) otimizado para transferência de arquivos em longas distancias. Este grupo também contém um serviço de localização de réplicas de dados. Também estão em desenvolvimento o projeto OGSA-DAI (*Open Grid Services Architecture Data Access and Integration*), que tem como objetivo construir um *middleware* que fará parte do Globus e que seja capaz de acessar e integrar fontes de dados através da grade[1] e um serviço de replicação de dados.

Gerência de Execução Este grupo é formado pelo GRAM (que será descrito a seguir). Em desenvolvimento existem o CSF (*Community Scheduler Framework*), um *framework* para o desenvolvimento de *metaschedulers* que operam em nível de organizações virtuais, o *Grid Telecontrol Protocol*, que permite o acesso remoto a instrumentos científicos e o *Workspace Management*, que permite o acesso a contas que o usuário possua em máquinas remotas.

Serviço de Informação. O *Web Monitoring & Discovery System* (MDS) fornece informações sobre recursos e serviços da grade. Estas informações são disponibilizadas para os usuários a partir de dados coletados pelo *Index Service*. O *Trigger Service* permite que sejam programadas ações a serem realizadas quando ocorrerem certos eventos, como por exemplo quando a utilização dos recursos do *site* atingir um determinado valor.

Elementos de tempo de execução. O Globus possui um núcleo para desenvolvimento de *web services* e clientes em conformidade com os padrões utilizados pelo *toolkit* em linguagens C, Java e Python.

O componente responsável pelo gerenciamento de recursos é o GRAM. Ele está presente em cada recurso (ou conjunto de recursos, no caso de *clusters* e MPP's) a ser gerenciado pelo Globus. Aplicações da grade precisam apenas se comunicar com o GRAM por meio de uma API (RSL – *Resource Specification Language*), e cabe ao módulo interagir com o escalonador local de cada recurso [32].

O GRAM é constituído por serviços genéricos que são traduzidos para chamadas dependentes de plataforma. Os serviços definidos pelo GRAM 4 são o *ManagedJob*, que fornece uma interface para as aplicações executando na grade (cada aplicação é traduzida em um *ManagedJob*) e o *ManagedJobFactory*, através do qual é possível coletar informações de estado das aplicações e solicitar o seu cancelamento.

O *ManagedJobFactory* é o serviço que representa os recursos da grade acessíveis através de um escalonador local. Este serviço fornece uma interface para criar recursos (*ManagedJobs*) a fim de permitir a execução de aplicações através de tal escalonador local. Portanto, se um determinado recurso permitir o acesso através de diferentes escalonadores locais, este recurso

[1] `http://www.ogsadai.org.uk`

oferecerá diversos serviços do tipo *ManagedJobFactory*, um para cada um dos escalonadores locais suportados.

Uma vez que o serviço *ManagedJob* é ativado no recurso remoto, um EPR (*End Point Reference*) é retornado ao usuário. Através deste EPR o cliente pode gerenciar a aplicação submetida, isto é, acompanhar o estado da aplicação, cancelá-la ou realizar uma operação de *attach* na aplicação. Esta última operação permite que o cliente possa ser informado sobre alteração no estado da aplicação e que possa receber diretamente a saída produzida por ela [37].

A descoberta de recursos é feita por agentes chamados *brokers*. Eles recebem as requisições em alto nível do usuário o podem traduzir em requisições mais específicas. Podem existir diversas "camadas" de *brokers*, cada uma refinando mais a requisição até que se chegue a requisições de recursos específicas.

A gerência de recursos pertencentes a diferentes *sites* que formam uma organização virtual pode ser desenvolvida com a utilização do CSF. Este *framework* provê um conjunto de *web services* que fornecem funcionalidades de reserva e submissão de aplicações. Usuários podem especificar os recursos desejados ou apenas os requisitos da sua aplicação, e o *metascheduler* encarrega-se do escalonamento. Para tanto, o *metascheduler* é capaz de interagir com os recursos via GRAM. A política a ser empregada na distribuição e escalonamento dos recursos, bem como outras funções adicionais (como por exemplo economia) deve ser implementadas pelo usuário [38].

O *toolkit* Globus não possui nem um *broker* nem um *grid scheduler* entre seus componentes. No entanto, existem diversas soluções desenvolvidas por terceiros, soluções estas apresentadas na Seção 2.2.

2.1.2 OurGrid

O OurGrid é um sistema de grades computacionais orientado ao sistema desenvolvido no Brasil. O OurGrid utiliza um conceito de *sites* em sua concepção [39, 40], e tem como foco aplicações do tipo BoT, embora versões recentes suportem a execução de aplicações paralelas que utilizam a biblioteca de troca de mensagens MPI.

A Figura 3 apresenta uma grade OurGrid. Ela é formada por um conjunto de *sites* que interagem entre si para obtenção de recursos. Usuários estão associados a um determinado *site*, e solicitam recursos para a execução de sua aplicação (através do *grid scheduler* MyGrid) ao gerente de recursos da grade local (chamado Peer). Estes recursos serão buscados, a princípio dentro do próprio *site*.

Em situações em que o Peer não seja capaz de fornecer aos usuários locais todos os recursos de que necessitam, ele envia pedidos de recursos a outros *peers* que compõem a comunidade. O modo como o pedido de recursos por *peers* remotos será atendido baseia-se em um modelo de *troca de favores* [39]: um Peer dará prioridade àqueles que mais lhe cederam recursos ante-

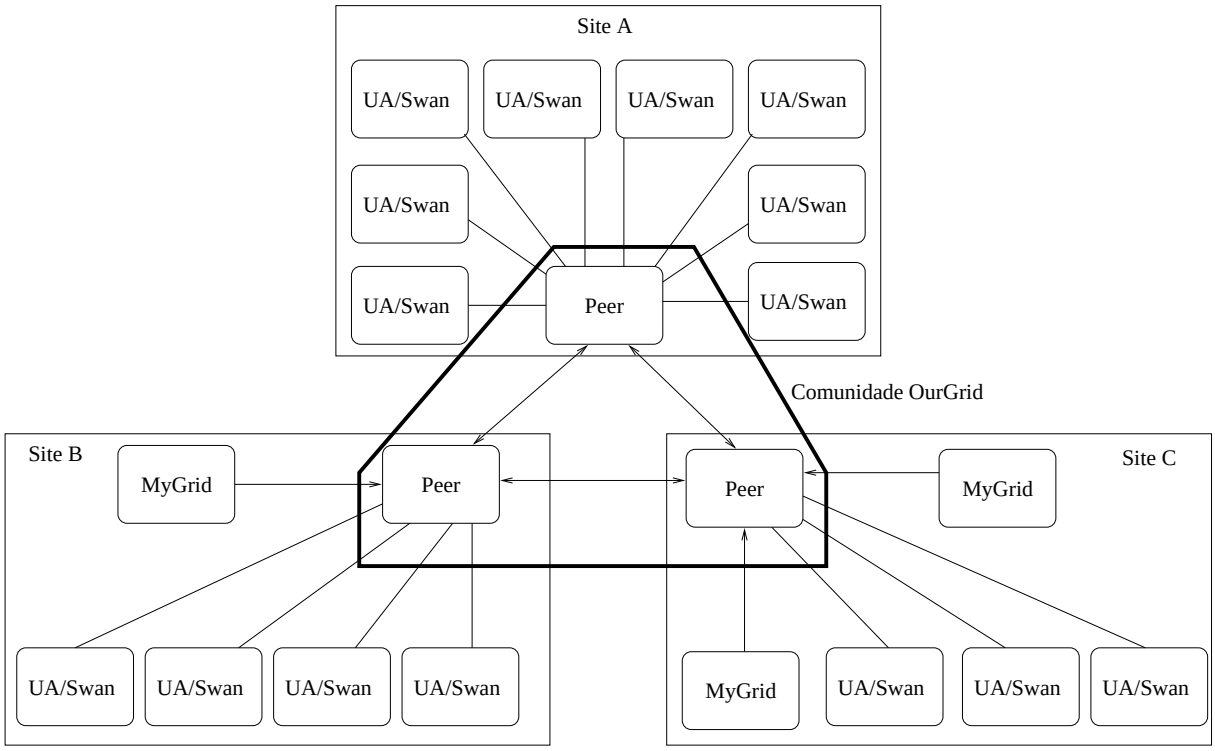

Figura 3 – A grade OurGrid.

riormente.

O principal aspecto a favor de um modelo econômico para grades baseado em troca de favores é a sua facilidade de implementação [39]. No OurGrid, a contabilidade dos favores é controlada localmente por cada Peer: cabe a ele manter uma tabela com cada um dos demais *peers* conhecidos por ele e o crédito de cada um. Estas tabelas locais são utilizadas apenas em caso de conflito por recursos. Isto torna possível que mesmo *peers* que nunca cedam recursos à grade obtenham favores, embora este comportamento tenda a fazer com que a probabilidade de recebê-los diminua.

É importante notar que os *peers* OurGrid não são *metaschedulers*: eles não realizam escalonamento, apenas encontram e adquirem recursos que posteriormente são delegados a clientes. O escalonamento das aplicações é realizada pelo MyGrid [15], um *grid scheduler* desenvolvido pela mesma equipe que desenvolve o OurGrid.

O ponto de partida na execução de uma aplicação na grade é uma máquina chamada *home machine*, a qual se supõe que o usuário tenha acesso para operação e instalação de *softwares*. Nela são instalados componentes do OurGrid responsáveis pelo escalonamento local da aplicação e pela interface com máquinas remotas pertencentes à grade (*grid machines*). As máquinas aptas a executar tarefas da grade devem executar um *software* chamado User Agent, que fornece uma interface uniforme para execução de aplicações e submissão de arquivos. No caso do OurGrid, é o Peer quem obtém máquinas de outros domínios, ou mesmo do seu próprio domínio, e as disponibiliza ao usuário. Após obtidas estas máquinas, elas passam a ser acessadas e controladas diretamente pelo usuário, através do *Processor Interface*.

Opcionalmente pode ser ativado no OurGrid um mecanismo de segurança, cuja implantação se dá através da configuração de cada um dos componentes (MyGrid, Peer e User Agent) para reconhecer e aceitar conexões seguras (através do protocolo TLS) dos demais componentes: no caso do MyGrid, ele precisa aceitar (armazenando sua chave pública e o incluindo em uma lista de confiança) o Peer ao qual solicita recursos. O Peer precisa reconhecer os User Agent dos recursos que controla, MyGrid de usuários pertencentes ao *site* e os demais Peers que pertencem à grade. User Agents só precisam aceitar conexões do Peer do seu *site*. Neste caso, o acesso a recursos de um *site* remoto por um usuário se dá através de uma cadeia de conexões seguras: MyGrid e Peer local, Peer local e Peer remoto, Peer remoto e User Agent.

Outro mecanismo oferecido pelo OurGrid é o Swan, que permite que User Agents executem em *sandboxes*, que evita acesso a componentes sensíveis do sistema, prevenindo desta forma que execuções de aplicações possam causar danos ao recurso.

2.2 Gerência e escalonamento de recursos em grades

A gerência de recursos em grades computacionais lida com a forma como serviços e recursos da grade podem ser disponibilizados a outras entidades [1]. No início das pesquisas em grades computacionais, a principal preocupação da comunidade de pesquisadores era a de desenvolver uma forma de prover acesso a recursos de forma uniforme. Desta etapa das pesquisas surgiram o GRAM – *Grid Resource Allocation and Management* – cuja função é prover uma forma de acesso a recursos da grade de forma uniforme, isto é, independente das características reais dos recursos e o OGSA (*Open Grid Services Architecture*) [7], que define a forma como serviços devem ser acessados.

Atualmente, são trabalhados em gerência de recursos aspectos coma a capacidade de formular acordos (SLA), isto é, protocolos para que serviços sejam oferecidos com algum tipo de qualidade de serviço [1] e escalonamento baseado em políticas de uso de recursos [41, 42].

O escalonamento em grades, sempre foi um tópico bastante pesquisado pela comunidade científica. Um dos prováveis motivos é que o Globus, padrão *de facto* em grades computacionais, não oferece *grid schedulers* entre os seus componentes, deixando esta questão em aberto para ser resolvida pela comunidade, que de fato tem desenvolvido diversos projetos deste tipo. Estes escalonadores atendem a necessidades específicas identificadas por seus desenvolvedores: alguns são desenvolvidos para serem utilizados com o Globus; outros, são capazes de interagir com o Globus e com outros sistemas de grades. Entre estes *grid schedulers* é possível destacar:

MyGrid. O MyGrid [15] é o *grid scheduler* do OurGrid. Ele escalona tarefas de aplicações BoT em recursos obtidos através de um Peer OurGrid. Os algoritmos de escalonamento utilizados são o *WorkQueue with Replication* (WQR) e o *Storage Affinity*. O WQR funciona da seguinte forma: cada uma das tarefas que compõem a aplicação do usuário é

distribuída para uma das máquinas da grade cedidas ao usuário. Se sobrarem máquinas para as quais não foram atribuídas tarefas, ou após todas as tarefas existentes já terem sido escalonadas e algumas máquinas já terem completado as suas tarefas, as tarefas ainda não completadas são replicadas nessas máquinas disponíveis. Tão logo uma das instâncias da tarefa se complete, as suas réplicas são canceladas. O *Storage Affinity* também trabalha com réplicas, da mesma forma que o WQR. A sua diferença é que, para decidir quais tarefas são enviadas para quais recursos, ele leva em conta a existência dos arquivos utilizados pelas tarefas nos recursos, e vai escalonar a tarefa preferencialmente naqueles recursos que irão exigir uma menor transferência de arquivos. Para realizar o armazenamento dos arquivos, o usuário determina explicitamente na execução das tarefas quais devem ser armazenados e quais serão utilizados apenas durante a execução da tarefa.

Condor-G. Este *grid scheduler* utiliza os protocolos do Globus (MDS e GRAM) para descoberta e acesso a recursos remotos e componentes herdados do sistema Condor [43] para gerência dos processos na grade. As tarefas do usuário são escalonadas entre os recursos disponíveis de acordo com algum critério fornecido pelo usuário. Como exemplo de critérios possíveis de serem utilizados estão o tempo esperado para início de execução, o tempo de término e o custo para alocação. Os recursos são ordenados de acordo com o critério e os processos são atribuídos obedecendo à fila de recursos. Caso os recursos disponíveis ao usuário possuam limitações quanto ao acesso a arquivos locais, o mecanismo de *GlideIn* pode ser usado. Este mecanismo faz com que os recursos não executem diretamente as aplicações do usuário, mas sim o *daemon* Condor. Com isto, é possível executar tarefas em *sandboxes* móveis que são disponibilizados pelo Condor. Estes *sandboxes* permitem que chamadas de sistemas executem na máquina do usuário ao invés de no recurso da grade e também permitem que o mecanismo de *checkpointing* do Condor seja utilizado [44].

Nimrod/G. Este é um *grid scheduler* que utiliza conceitos de economia de grade para suas decisões de escalonamento [45, 46]. Suporta aplicações do tipo *parameter sweep* e possui módulos que o tornam compatível com o Globus e o Legion, embora módulos para outros sistemas possam ser desenvolvidos pelos usuários. Inclui componentes para supervisão do sistema, controle de aplicações e escalonamento, podendo este ser realizado de duas formas. Na primeira procura encontrar recursos dentro do limite de preço que o usuário se propõe a pagar e dentro do prazo para execução da aplicação (também definido pelo usuário). Na segunda forma de escalonamento o usuário negocia os recursos que irá utilizar, estipulando o valor que pagará para que a tarefa se complete dentro do prazo. Caso o prazo não possa ser cumprido com o orçamento estipulado, um novo prazo ou um novo valor devem ser negociados. As tarefas que compõem a aplicação do usuário são geradas automaticamente pelo *grid scheduler* a partir de uma descrição de alto nível da aplicação. O Nimrod/G utiliza os serviços de segurança e informação existentes nos

middleware de grade.

GridWay. O GridWay é um *framework* para escalonamento, monitoração e gerência de aplicações *parameter sweep* para ser utilizado em grades Globus. O seu diferencial com relação aos demais sistema é a capacidade de realizar migração de tarefas [47]. As tarefas podem migrar para compensar perda de recursos (devido a falhas ou cancelamento de tarefas ou devido a decisões administrativas) ou oportunisticamente (para aproveitar recursos "melhores" que tenham se tornado disponíveis ou quando o recurso executando a tarefa não apresentar um desempenho satisfatório). O GridWay suporta ainda *checkpointing* em nível de aplicação, através do geração de arquivos de reinício. Na ausência deste arquivo, uma tarefa cancelada irá executar desde o início. Para classificar os recursos disponíveis ao usuário, a fim de determinar para quais recursos preferencialmente serão submetidas as tarefas, este deve fornecer uma expressão que combina os diversos parâmetros de um recurso que são fornecidos pelo Globus MDS [48].

A vantagem da utilização de *grid schedulers* é a possibilidade de abstrair a descoberta e o escalonamento de recursos em grades. Ao utilizar estas ferramentas, usuários têm acesso ainda a funcionalidades que reduzem o custo de utilização de recursos (no caso do Nimrod/G) ou que melhoram o desempenho das aplicações, como algoritmos de escalonamento otimizados para determinadas classes de aplicações (presentes no MyGrid), mecanismos de migração de tarefas (no caso do GridWay) e checkpointing (presente no GridWay e no Condor-G). A não utilização deste sistema implica em escalonamento de recursos manual por parte do usuário, o que conflita com a questão de transparência de grades. Por estas razões, a utilização destas ferramentas deve ser considerada por usuários de grades.

3 O Site Resource Scheduler

Nesta seção será apresentada uma proposta de arquitetura e os serviços suportados pelo *Site Resource Scheduler*. O SRS pode ser implantado como um *middleware* específico, caso em que deveriam ser desenvolvidos todos os módulos no *site* e também o *broker* do usuário.

Alternativamente, o SRS pode ser implantado como um módulo em algum *middleware* existente. Neste caso, os módulos que interagem com clientes e outros *sites* (caso exista esta comunicação) devem ser mantidos sem alteração, para não afetar a compatibilidade com outros *sites* que não utilizam o SRS, e todas as questões relevantes para este tipo de *software* (segurança, autenticação de usuários, tolerância a falhas, economia e demais características descritas na Seção 1.1) poderiam ser abordadas. No entanto, o foco deste capítulo será na questão da virtualização dos recursos do *site*, e os demais aspectos serão apenas apresentados e brevemente discutidos.

3.1 Motivação

Nos *middleware* para grades computacionais atuais, os usuários da grade[1], diretamente ou através de *brokers*, são responsáveis pela descoberta, alocação e gerência de recursos a eles destinados.

Quando um cliente possui sob seu controle máquinas que podem estar distribuídas por todo o mundo (Figura 4), a gerência destes recursos introduz um *overhead* na execução da sua aplicação.

Este problema é ainda mais crítico quando os recursos fornecidos ao usuário não são dedicados: neste caso, além do ônus da gerência de recursos realizado pelo usuário através do seu *grid scheduler*, é necessária uma monitoração dos recursos e escalonamento de aplicações quando recursos forem removidos da grade. O novo escalonamento atualmente exige uma nova submissão de arquivos de entrada e nova ordem de execução de tarefas.

Se o novo escalonamento das tarefas for para uma máquina pertencente ao mesmo *site* da máquina que o cliente perdeu, a transferência de arquivos de entrada poderia ser evitada se arquivos de clientes pudessem ser temporariamente armazenados no *site*. Uma alternativa ainda melhor seria que a própria perda do recurso e conseqüente escalonamento da aplicação fosse gerenciada pelo *site* de forma transparente ao cliente: desta forma, do seu ponto de vista a submissão de arquivos e execução ocorre em um recurso que está disponível a ele durante todo

[1] No restante deste capítulo o termo “cliente” será usado como sinônimo para usuários da grade.

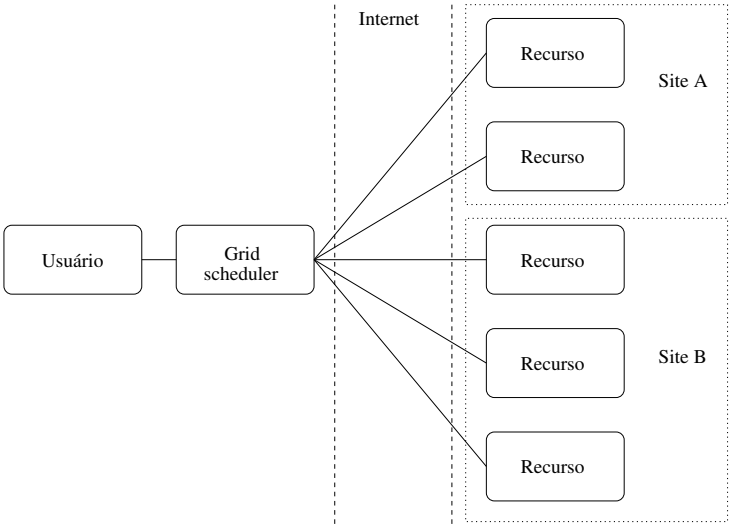

Figura 4 – Modelo atual de escalonamento de recursos em grades.

o tempo de execução da aplicação e que não é retirado da grade para priorizar usuários locais.

3.2 Um novo modelo de escalonamento em grades

O modelo de escalonamento na grade proposto neste trabalho possui um conjunto de características que difere dos modelos disponíveis atualmente. A Figura 5 ilustra a abordagem, que possui as seguintes características:

- A visão que um cliente da grade tem dos recursos de um *site* muda de um conjunto de máquinas reais com determinada capacidade para o de uma única máquina com uma capacidade computacional que foi previamente negociada, variando em função da carga atual do *site* e das permissões do usuário na utilização dos recursos;

- O escalonamento, antes realizado completamente pelo usuário passa a ocorrer em dois níveis, com o usuário submetendo tarefas para *sites* e estes escalonando as tarefas recebidas entre os recursos locais;

- A gerência dos recursos ocorre dentro do *site*, onde pode haver uma resposta mais rápida a eventos relacionados aos recursos;

- O tratamento de indisponibilidade de recursos pode ocorrer de uma forma mais eficiente, pois neste caso um novo recurso dentro do *site* poderá ser disponibilizado, não sendo necessária uma nova fase de descoberta de recursos e submissão de arquivos por parte dos usuários;

- A escalabilidade dos *grid schedulers* aumenta, pois a quantidade de recursos gerenciada por eles passa de um grande número de máquinas para uma menor quantidade de *sites*.

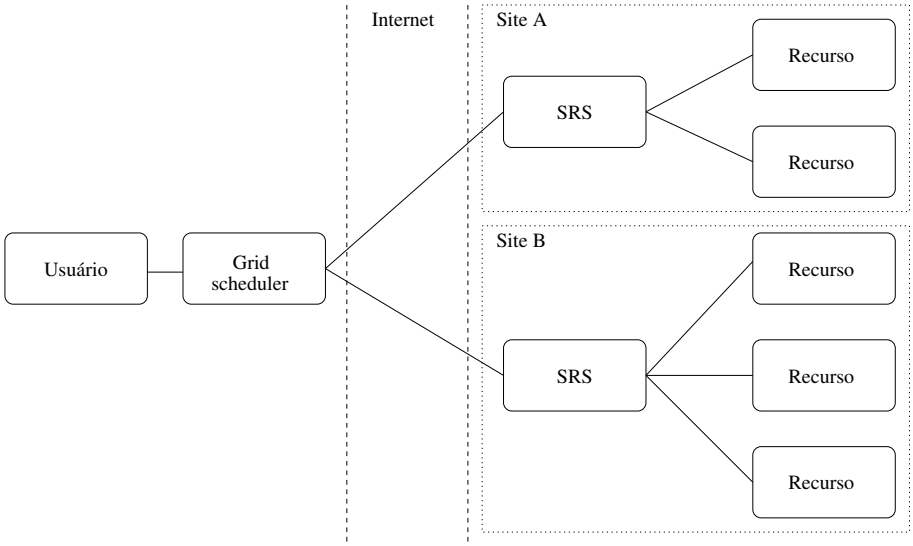

Figura 5 – Modelo proposto para o escalonamento em grades.

Eventualmente, os recursos gerenciados por um SRS poderiam ser outros SRS's, provendo-se desta forma uma estrutura hierárquica que aumentaria ainda mais a escalabilidade do sistema. A forma de fornecer esta visão a usuários é através da virtualização dos recursos do *site*, conforme será apresentado na seção seguinte.

3.3 Virtualização

Propõe-se como meio para obtenção das características apresentadas na seção anterior a virtualização dos recursos de um *site*, de forma que usuários não tenham acesso a recursos reais e sim a recursos virtuais que escondem características específicas dos recursos, estando incluídas nestas características o número de máquinas disponíveis.

Para que este nível de virtualização seja obtido, é necessário que requisições de clientes e entrega de recursos não aconteçam na forma de máquinas específicas: a enumeração de máquinas do *site* deve ser substituída por uma unidade na qual o cliente possa expressar a sua necessidade computacional. O *site* deve ser capaz de traduzir esta unidade abstrata em um número real de recursos disponíveis. Estes recursos não devem ser monopolizados por um usuário, portanto o *site* também deve ser capaz de distribuir os recursos entre os seus diversos usuários.

A seguir serão apresentadas as duas principais questões relacionadas a virtualização dos recursos de um *site*: a determinação da capacidade computacional do *site* e a forma com que os recursos podem ser distribuídos de forma a atender a demanda de capacidade computacional dos usuários.

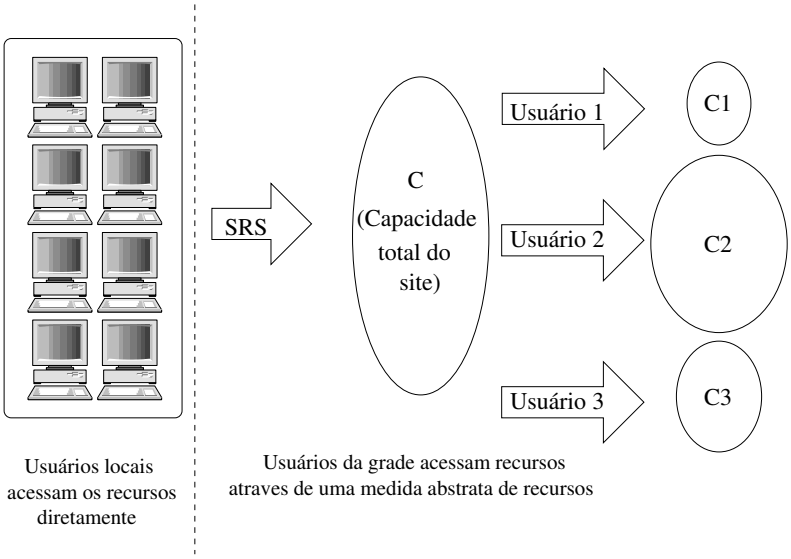

Figura 6 – Virtualização dos recursos do *site*.

3.3.1 Determinação da capacidade computacional

O objetivo da virtualização proposta neste trabalho é esconder características particulares dos recursos do *site* que os cede. Ao mesmo tempo, usuários locais devem ter acesso aos recursos reais do *site*. A Figura 6 ilustra este fato. Do ponto de vista dos usuários locais, recursos do *site* são acessados diretamente. Usuários da grade acessam os recursos através do SRS, que os apresenta na forma de uma capacidade computacional. Porém, a quantidade de capacidade computacional apresentada para cada usuário pode ser diferente, em função de características específicas necessárias às suas aplicações ou em virtude de seus direitos de acesso.

Quanto a forma com que os recursos serão representados no *site*, busca-se aquela mais abstrata possível, isto é, aquela que exige um menor detalhamento das características específicas do recurso. Porém, a aplicação de uma medida abstrata depende também da forma como o *grid scheduler* escalona tarefas aos recursos: se ele despacha tarefas para os recursos seqüencialmente, uma alta abstração não será vantajosa, pois como a visão do usuário propiciada pelo SRS é de uma única máquina com grande capacidade, o SRS receberá uma única tarefa de cada vez, e mesmo que existem no momento mais máquinas disponíveis ao usuário, elas não serão utilizadas.

Para casos semelhantes, a unidade de capacidade computacional disponível para o usuário a ser empregada é a quantidade de “máquinas virtuais” disponíveis ao cliente. Estas máquinas não mapeiam diretamente nenhum recurso específico: em um determinado momento, uma máquina virtual pode estar mapeada em zero (neste caso, a tarefa do usuário está na fila do escalonador do SRS) ou uma máquina real. Se o escalonador do usuário for capaz de despachar mais de uma tarefa simultaneamente ao recurso, também é possível que um recurso virtual mapeie mais de um recurso real. Neste caso, cada um dos recursos reais recebe uma tarefa para execução.

Para casos em que o *grid scheduler* é capaz de dividir tarefas proporcionalmente a capacidade computacional de cada recurso disponível, uma unidade mais abstrata pode ser utilizada. A unidade proposta, é baseada em um método utilizado por Xiao et al. [49] e utiliza *benchmarks* para determinar a capacidade dos recursos.

Na proposta de Xiao, cada recurso é avaliado com a utilização do *benchmark* SPEC[2]. É utilizada então como medida de capacidade do recurso a sua capacidade avaliada em relação a de uma plataforma padrão, ou seja, tem-se uma medida de quantas vezes o recurso disponível é melhor (ou pior) que o recurso padrão para aquele teste.

Como medida de capacidade computacional do SRS, propõe-se uma idéia semelhante (a utilização do desempenho relativo a uma plataforma padrão) porém com o *benchmark* compatível com o tipo de operação utilizada pela aplicação do usuário: se a aplicação utilizar basicamente operações de números inteiros, deve ser utilizado um *benchmark* que avalia operações com números inteiros e se a aplicação for predominantemente composta de operações de ponto flutuante, deve ser utilizado um *benchmark* que avalia este tipo de operação. Um exemplo do primeiro tipo é o Compress do pacote de *benchmarks* SPEC92 [50] e do segundo é o Alvin do mesmo pacote. Cada máquina do *site* deve ter seu desempenho medido em cada um dos *benchmarks* e comparado com a máquina padrão no mesmo teste. Estes valores devem ser armazenados pelo SRS para serem utilizados quando houverem requisições de recursos.

Para o usuário determinar qual a capacidade computacional requerida pela sua aplicação, esta deve ser executada sobre a plataforma padrão, e este tempo deve ser utilizado no cálculo da capacidade necessária. A demanda do usuário é medida então como o tempo de utilização de um recurso da plataforma padrão. Se não houver a possibilidade de obter este tempo diretamente (isto é, utilizando efetivamente um recurso da plataforma padrão), ele pode ser determinado indiretamente, utilizando-se um recurso qualquer e aplicando sobre o tempo obtido um fator que é a capacidade do recurso em relação à plataforma alvo.

Por exemplo, se uma aplicação é executada em 40 minutos em uma máquina que executa o *benchmark* apropriado duas vezes mais rápido que a máquina padrão (ou seja, a máquina é duas vezes melhor que a padrão), então a necessidade do usuário é de 80 “minutos”.

É importante salientar que outros fatores devem ser levados em conta por clientes descrevendo os seus requisitos e pelo SRS: clientes devem incluir nos seus requisitos, juntamente com a capacidade computacional requerida, alguma característica exigida pela sua aplicação: a aplicação pode funcionar somente sobre um sistema operacional ou plataforma específica, ou exigir um mínimo de memória RAM ou ainda necessitar de um *software* especial[3]. Para calcular a capacidade computacional disponível ao usuário, devem ser considerados todos os requisitos do usuário e mais aspectos particulares do *site*: ocupação atual dos recursos, direitos de acesso dos usuários e, eventualmente, expectativa de carga futura.

[2] `http://www.specbench.org`

[3] Este tipo de especificação é comum em grades, sendo necessária tanto no OurGrid quanto no Globus.

3.3.2 Distribuição de recursos entre usuários

Um segundo aspecto importante para a implementação da virtualização proposta é a questão da distribuição de recursos. Esta questão ganha relevância no SRS porque, diferente de outros sistemas de grades, o usuário não recebe e gerencia os recursos diretamente. Por isso, cabe ao SRS a tarefa de garantir uma determinada quantidade de recursos ao usuário, fazendo com que a execução da sua tarefa ocorra o mais próximo possível dentro do tempo compatível com a capacidade computacional oferecida.

Esta tarefa não é trivial pois deve considerar alguns fatores. Um deles é o fato que usuários locais têm prioridade sobre a utilização dos recursos e acesso direto a eles (os recursos não são virtualizados dentro do *site*). Isto pode prejudicar algum usuário da grade cujas tarefas se encontram em máquinas que são requeridas diretamente por usuários locais.

A manutenção de histórico de utilização de recursos pode ajudar a minimizar efeitos negativos ocasionados por estes fatos: o escalonador do SRS pode evitar mapear todas as tarefas de um cliente em recursos com alta taxa de utilização e/ou em máquinas que possuem uma alta probabilidade de serem requisitados localmente dentro do período estimado de execução da sua tarefa.

Outro aspecto a ser considerado é a forma com que recursos são distribuídos entre usuários da grade. Embora o SRS deva calcular a capacidade computacional disponível para um cliente baseado em seus direitos de acesso e carga atual do *site*, o SRS não deve oferecer toda a capacidade disponível mesmo que o usuário tenha direito de acesso a todos estes recursos.

O objetivo disto é garantir que mais usuários consigam executar suas aplicações, aumentando o desempenho global da grade, à custa da diminuição do desempenho individual de usuários.

Outra forma possível de garantir distribuição de recursos é aplicando alguma política de preempção e redistribuição de recursos de forma transparente ao usuário. Esta afirmação é válida principalmente para aplicações do tipo *parameter sweep* e BoT, pois nestes modelos de programação as tarefas são independentes, e a interrupção de uma não influi no desenvolvimento das demais tarefas.

Uma forma simples de equilibrar o uso de recursos é garantir, através de preempção de recursos, que usuários com direitos equivalentes possuam a mesma quantidade de recursos, podendo esta comparação ser feita em número de máquinas ou ainda na capacidade computacional medida de cada máquina. Para isto, o SRS pode cancelar tarefas de um usuário e submeter tarefas de outro usuário nos recursos anteriormente utilizados pelo primeiro, de forma transparente aos usuários.

Uma maneira mais eficiente de redistribuir recursos pode ser obtida se as aplicações (ou o SRS) implementarem *checkpointing* [51]. Neste caso, pode ser aplicada uma política de escalonamento de tempo compartilhado, semelhante àquela aplicada em sistemas operacionais

multitarefa. Neste caso, devem ser buscados valores para duração da fatia de tempo e número de tarefas por máquina que tornem a aplicação desta política atraente tanto para clientes que executam tarefas de grande duração quanto para os que executam tarefas de pequena duração. Após a execução de uma tarefa por um tempo, ela é temporariamente interrompida para dar lugar a execução de uma outra tarefa. O mecanismo de *checkpointing* permite que a execução da tarefa seja retomada do ponto em que ela parou quando preemptada e, com a utilização de mecanismos de migração de tarefas, em uma máquina diferente daquela onde a execução iniciou.

3.4 Arquitetura

Esta seção apresenta uma visão geral da arquitetura do *Site Resource Scheduler*. O SRS é composto por módulos internos, cada um realizando serviços relacionados a um determinado aspecto da computação em grade. A interação entre estes módulos é representada na Figura 7. Os módulos identificados foram:

- Um *módulo de informação*, responsável por armazenar informações sobre o *site* e seus usuários. Informações relevantes sobre o *site* incluem aquelas relacionadas aos recursos e suas capacidades e aos usuários e suas permissões. Estas informações são úteis para melhorar a qualidade do escalonamento e garantir o cumprimento de políticas locais de utilização de recursos;
- Um *módulo de interface de usuário*, que recebe todas as requisições de usuários e as encaminha ao módulo apropriado;
- Um *módulo de gerência de recursos*, que controla todos os recursos do *site*, atribuindo aqueles disponíveis à grade ao módulo de escalonamento e provendo informações ao módulo de informação;
- Um *módulo de escalonamento*, que recebe tarefas (submetidas através do módulo de interface de usuário), recursos disponíveis (recebidos através do módulo de gerência de recursos) e realiza o mapeamento e execução das tarefas. A heurística aplicada (se utilizada) pode ser determinada pelo administrador do *site* através de um arquivo de configuração;
- Um *módulo de gerência de arquivos*, responsável pela gerência dos arquivos submetidos ao SRS. Este módulo também aplica a política de cotas utilizada no *site*, quando existir;
- Um *módulo de segurança*, responsável por aplicar protocolos de segurança no *site*;
- Um *módulo de economia*, responsável pela contabilização da utilização dos recursos do *site* e aplicação da política de cobrança por estes recursos estipulada pelo seu administrador.

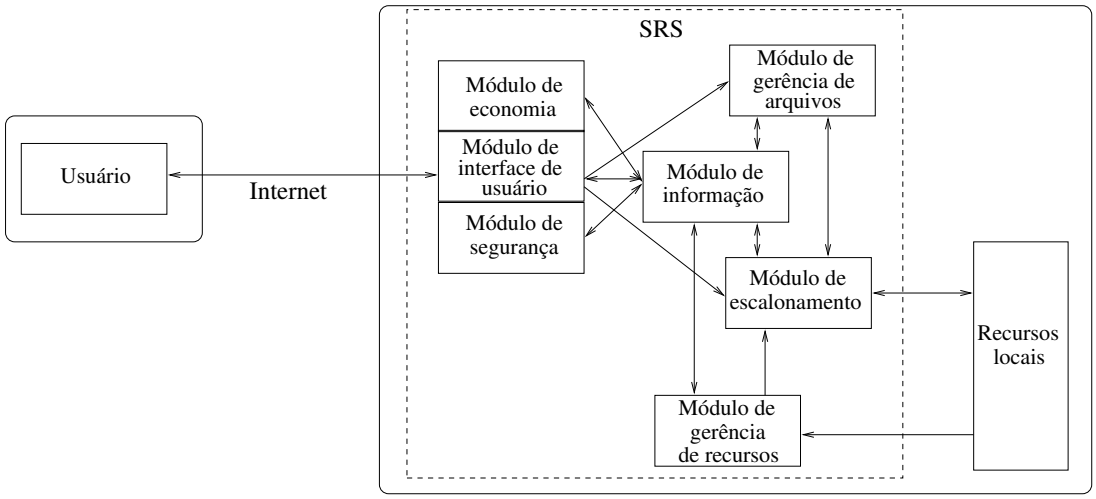

Figura 7 – Organização modular do SRS.

Os módulos de interface de usuário e de escalonamento são módulos que implementam os principais aspectos que diferenciam o SRS dos sistemas existentes. O primeiro exterioriza a virtualização realizada pelo sistema, enquanto o segundo aplica o escalonamento sobre os recursos da grade, o que inclui a alocação não-dedicada de *clusters*. Por isso, eles serão descritos mais detalhadamente a seguir, após uma explicação sobre como se dá a execução de aplicações no SRS.

3.4.1 Execução de aplicações no SRS

Nesta seção é descrito o protocolo de interação entre clientes e SRS. Estes passos estão ilustrados no diagrama de seqüência [52] da Figura 8.

O primeiro passo é a consulta ao *Site Resource Scheduler* sobre a capacidade computacional atualmente disponível ao cliente. Esta etapa inicia com a identificação do cliente (através da apresentação de uma credencial ou qualquer outro método utilizado no *site* específico). Neste passo também devem ser descritos os requisitos do usuário.

O SRS então confere as credenciais do cliente e, baseado em seus direitos de acesso, na carga atual do *site* e nos requisitos da aplicação, é apresentada ao cliente a capacidade computacional disponível, e caso se aplique, o preço pelo seu uso. Este passo pode ser realizado pelo cliente sobre diversos *sites* simultaneamente.

O cliente decide então em que *sites* ele deseja executar a sua aplicação. Esta decisão é tomada por ele e pode envolver critérios tais como preço, quantidade de capacidade disponível e presença de arquivos necessários à aplicação. Posteriormente, deve ser feita a requisição de capacidade computacional (de forma dedicada ou não-dedicada) em uma quantidade que pode ser igual ou menor àquela oferecida pelo SRS, de acordo com a decisão do escalonador. A principal razão para usuários não requisitarem a totalidade de recursos disponíveis para eles é

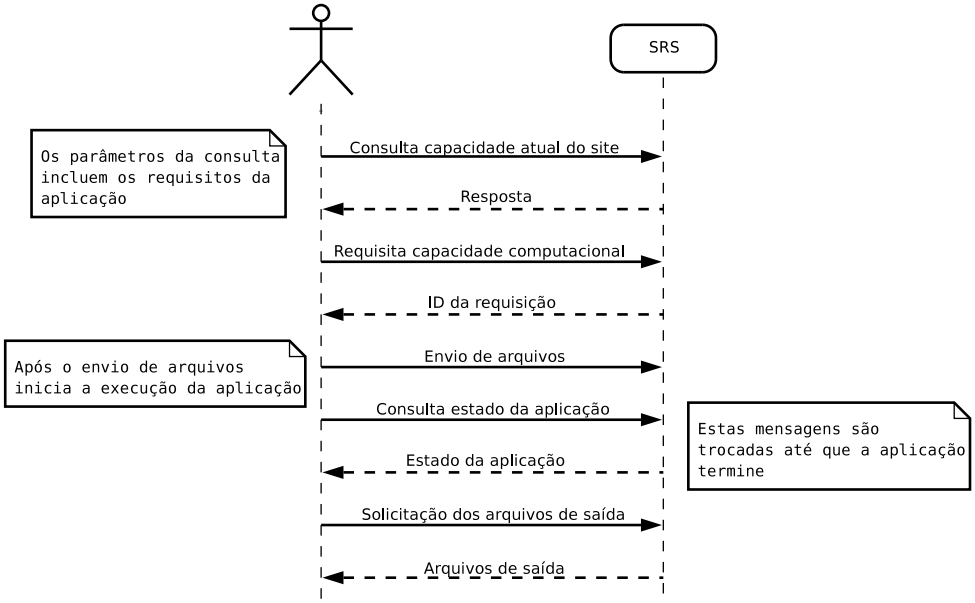

Figura 8 – Diagrama de seqüência: interação entre cliente e SRS durante a requisição de execução de aplicações.

que, caso seja adotada uma política de contabilização de uso de recursos, usuários poderiam economizar créditos pela sua utilização.

Após decidida a quantidade de capacidade alocada de cada *site*, o escalonador local divide as aplicações de acordo com a capacidade requerida de cada *site*. Aplicações são submetidas através de uma especificação que deve incluir:

- Arquivos que devem ser transferidos para o SRS;
- Método de armazenamento (temporário ou persistente) de cada arquivo;
- Linha de comando da aplicação a ser executada;
- Método de armazenamento que deve ser aplicado sobre cada arquivo gerado pela aplicação.

Este passo é encerrado com o fornecimento da identificação da aplicação ao cliente, e após isso os arquivos são transferidos e a aplicação é executada no *site*. Clientes podem consultar o SRS sobre o estado destas aplicações e, ao seu término, podem solicitar o *download* dos resultados.

Outros serviços, relacionados à gerência de arquivos e contabilidade podem ocorrer a qualquer momento, mas precisam ser precedidas de uma fase de autenticação, tal como aquele presente no início do processo de execução de aplicações.

3.4.2 O módulo de interface de usuário

O módulo de interface de usuário é o módulo que serve de entrada de requisições de usuários no sistema. Sua interface contém métodos que refletem os serviços disponibilizado pelo SRS aos clientes e também métodos através dos quais outros módulos podem fornecer notificações de eventos.

Além de fornecer interfaces para cada um dos serviços oferecidos aos clientes (apresentados na seção seguinte), este módulo deve oferecer uma interface para o módulo de escalonamento, através do qual o módulo de interface de usuário recebe informações de estado das aplicações em execução. O diagrama de seqüência da Figura 9 mostra como este módulo interage com clientes e demais módulos durante a execução de aplicações. Para melhorar a clareza da figura, nela não está representada a etapas de consulta ao estado da aplicação.

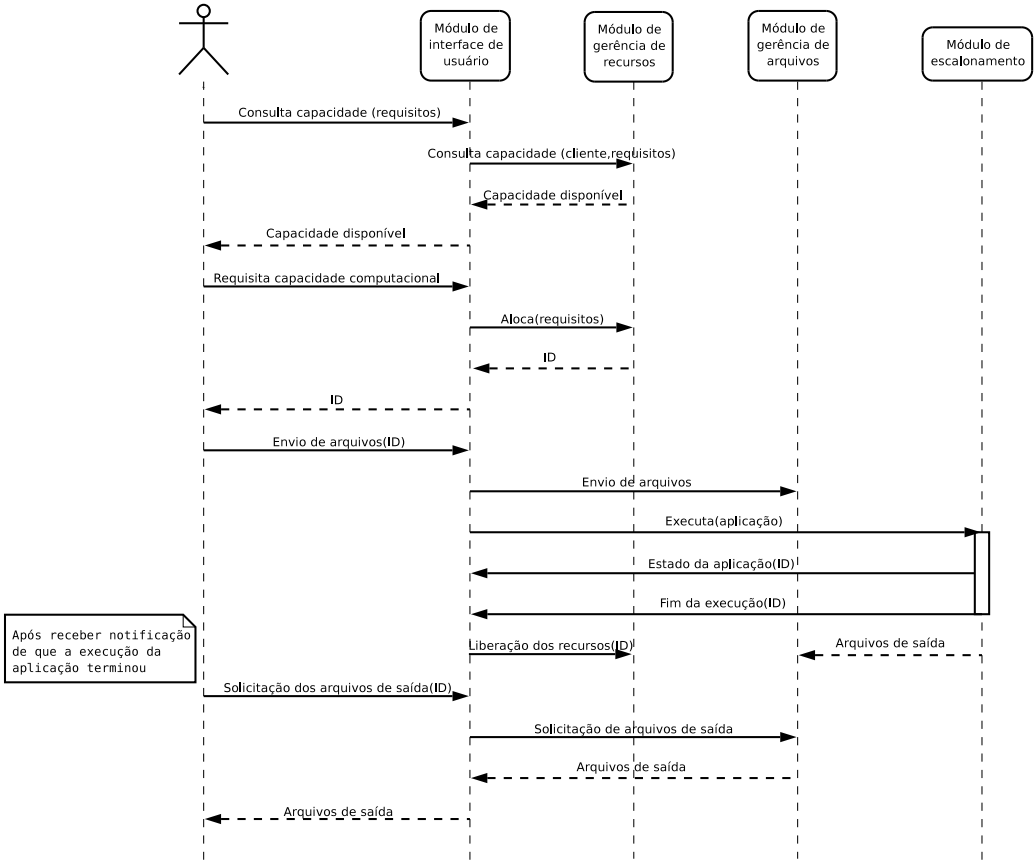

Figura 9 – Diagrama de seqüência: interações do módulo de interface de usuário durante a execução de aplicações.

Um serviço do módulo de gerência de recursos cujo objetivo é realizar a alocação dos recursos é invocado. Se a alocação for do tipo dedicada, os recursos são efetivamente alocados e ficam a disposição do cliente para a execução da aplicação. De qualquer forma, uma identificação é retornada pelo módulo de gerência de recursos. Esta identificação é repassada ao cliente pelo módulo de interface de usuário.

Quando o usuário submete os arquivos, estes são enviados ao módulo de gerência de arquivos. Após todos os arquivos serem recebidos, a aplicação é submetida ao escalonador e a execução pode começar. A execução da aplicação envolve a transmissão dos arquivos de entrada para a máquina que executará a aplicação, a execução e a submissão dos resultados ao módulo de gerência de arquivos para que esteja disponível ao cliente quando este os requisitar.

O término da execução é comunicado pelo módulo de escalonamento ao módulo de interface de usuário, através da identificação da aplicação. A partir do momento que a notificação é recebida, uma consulta do usuário pelo resultado da aplicação retornará a informação e solicitações por arquivos de saída são atendidas através de solicitação ao módulo de gerência de arquivos.

3.4.3 O módulo de escalonamento

O módulo de escalonamento é o módulo que, a partir de um conjunto de tarefas recebidas pelo módulo de interface de usuário, um conjunto de recursos recebidos pelo módulo de gerência de recursos, informações extraídas pelo modulo de informações e de uma heurística de escalonamento escolhida pelo administrador do sistema, mapeia as tarefas em recursos, controlando a sua execução.

O módulo de informações é consultado para que informações sobre prioridades de execução e forma de alocação de recursos possam ser utilizados durante o escalonamento, se a heurística aplicada fizer uso destas informações. O módulo de informações também pode fornecer informações sobre os recursos que podem ser utilizadas por heurísticas de escalonamento.

O funcionamento geral do módulo é apresentado no diagrama de atividade [52] da Figura 10.

Aplicações recebidas do módulo de interface de usuário são armazenadas, divididas em tarefas e, antes de serem despachadas para execução, ocorre a consulta sobre os direitos de acesso do usuário que solicitou a execução. As tarefas são encaminhadas para uma fila e são mapeadas baseadas em critérios estabelecidos pela heurística utilizada.

O módulo de escalonamento também é informado de mudanças nos estados dos recursos do *site*, isto é, quando os recursos tornam-se disponíveis para o SRS e quando são delegados a usuários locais, momento em que são retirados da grade.

Quando uma tarefa vai ser executada, é gerada uma *thread* para controle da execução da tarefa. Esta *thread*, chamada de *monitor de aplicação* é responsável por:

- Transferir arquivos do módulo de gerência de arquivos para o recurso;
- Disparar a aplicação no recurso e monitorar o seu estado;
- Após a execução da aplicação, transferir os arquivos de saída do recurso para o módulo de gerência de arquivos.

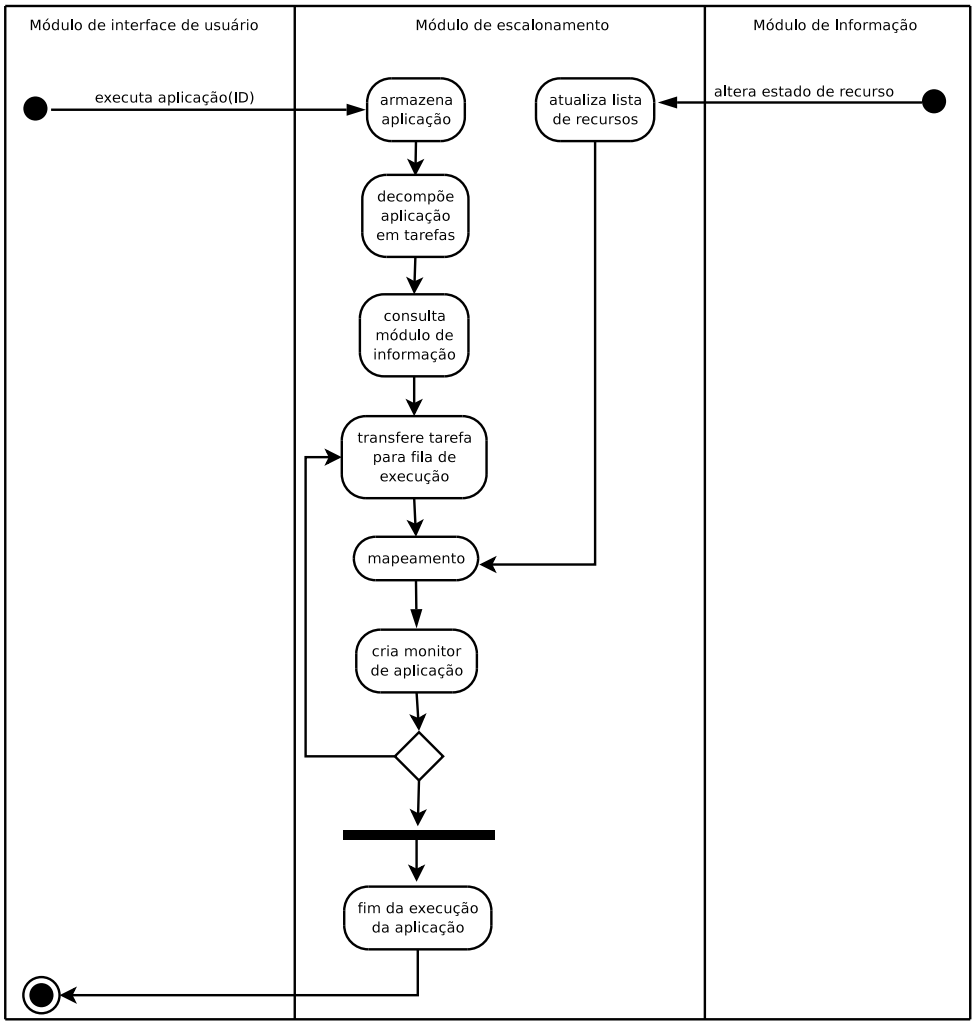

Figura 10 – Diagrama de atividade: funcionamento do módulo de escalonamento.

O monitor de aplicação pode terminar a sua execução com sucesso devido (i) ao fato do recurso ter sido removido da grade ou (ii) porque a tarefa foi corretamente executada e os arquivos de saída transferidos. Caso o monitor de aplicação encerre devido ao primeiro caso, a tarefa deve retornar para a fila de execução. Caso o monitor de aplicação encerre devido ao segundo caso, a tarefa é marcada como terminada. Após o término de todas as tarefas, a aplicação é completada.

Se as tarefas que compõem a aplicação forem fracamente acopladas, poderão ser distribuídas entre os recursos disponíveis para o usuário e executadas em qualquer ordem, isto é, se alguma das tarefas for interrompida, a tarefa retorna para a fila de execução e aguarda um recurso disponível. Caso exista alguma relação de ordem na execução das tarefas, esta deve ser respeitada pelo escalonador. Convém salientar que diferentes heurísticas de escalonamento gerarão desempenhos diferentes na execução das aplicações, dependendo do seu tipo. A escolha da heurística (ou heurísticas) de escalonamento a ser empregada no *site* deve levar este fator em consideração.

Interação com escalonadores locais

Por ser capaz de gerenciar todos os recursos de um *site*, o SRS, através do módulo de escalonamento, também deve interagir com máquinas de espaço compartilhado, tal como um COW. Estas máquinas são geralmente gerenciadas por escalonadores que coordenam o acesso à máquina, gerenciam filas de execução e proporcionam uma utilização eficiente do recurso [53]. Alguns exemplos de escalonadores de *clusters* são CCS [54], SLURM [55] e CRONO [56].

O SRS deve ser capaz de comunicar-se com os escalonadores de máquinas paralelas a fim de obter recursos para os clientes da grade. Eventualmente, o SRS deverá ser capaz de alocar recursos através destes escalonadores, e na maioria dos escalonadores isto implica na determinação de uma quantidade de máquinas e de tempo pelo qual se deseja utilizar os recursos. Alternativamente, podem ser aplicadas técnicas de alocação transparente sobre os recursos paralelos [57]. Neste caso, as máquinas só estarão a disposição do SRS quando não estiverem sendo utilizadas por nenhum usuário do *cluster*, e os recursos serão preemptados da grade quando for recebido um pedido de alocação de um usuário local.

3.5 Serviços

Nesta seção é apresentado o conjunto de serviços oferecidos pelo *Site Resource Scheduler* a clientes da grade. Neste contexto, são considerados clientes escalonadores e *brokers* a serviço de aplicações de usuários, usuários de sistemas de grade que não realizam escalonamento automático ou *metaschedulers*. Estes serviços podem ser divididos em cinco grupos:

- Gerência de recursos;
- Gerência de aplicações;
- Gerência de arquivos;
- Segurança e controle de acesso;
- Economia.

A seguir são descritos os serviços pertencentes a cada uma destas categorias.

3.5.1 Gerência de recursos

Este grupo contém serviços relacionados à interação entre clientes e o SRS. Entre estes serviços estão incluídos descoberta de capacidade computacional do *site*, reserva e alocação de recursos e execução, monitoração e cancelamento de aplicações.

Determinação da capacidade computacional do site. Através deste serviço, clientes podem descobrir a quantidade de poder computacional que o *site* pode fornecer no momento. Este poder pode ser fornecido em qualquer uma das unidades discutidas anteriormente. Este serviço apenas fornece a informação requisitada, não realizando qualquer alocação sobre os recursos ou garantia sobre a manutenção deste valor no futuro. Isto é feito pelos três serviços descritos a seguir.

Alocação dedicada de recursos. Através deste serviço usuários podem requisitar uma quantidade de poder computacional que será fornecida na forma de recursos dedicados, tais como *clusters* e MPP's [3]. Este serviço é fornecido para ser utilizado em aplicações que necessitem de um número fixo de máquinas, como por exemplo aplicações MPI. A utilização deste serviço deve implicar em uma cobrança maior do usuário, pois este serviço exige maior disponibilidade dos recursos do *site*, uma vez que neste caso recursos não serão preemptados do usuário.

Alocação não-dedicada de recursos. Através deste serviço usuários solicitam uma quantidade de capacidade computacional que será fornecida através de recursos não-dedicados, como por exemplo estações de trabalho ociosas. Este serviço pode ser utilizado em aplicações fracamente acopladas e em qualquer tipo de aplicação que suporte *checkpointing*. Tarefas canceladas em recursos que se tornam indisponíveis para a grade são escalonados em outra máquina disponível ou então são incluídas na fila de espera do escalonador do SRS, que é invisível para o usuário (isto é, do ponto de vista do usuário sua aplicação continua executando). É importante salientar que alocação não-dedicada de recursos pode ser realizada também sobre *clusters* [57].

Reserva de recursos. Este serviço fornece uma quantidade de poder computacional que será disponibilizado em um momento futuro. Esta reserva pode ser feita para alocações dedicadas e não-dedicadas.

Requisição adicional de recursos. Este serviço habilita usuários a receberam uma quantidade de recursos superior aquela concedida inicialmente, respeitando os seus limites de acesso.

3.5.2 Gerência de aplicações

Os serviços desta categoria são aqueles relacionados a execução de aplicações de clientes.

Execução de aplicação. Este é o serviço utilizado pelos clientes quando desejam executar uma aplicação no SRS. É preciso ser fornecido ao SRS o tipo de aplicação sendo executada, seus parâmetros e arquivos que precisam ser fornecidos na entrada e na saída. O SRS responde

à solicitação com uma identificação única representando a aplicação submetida. Esta identificação é utilizada em consultas sobre o estado da aplicação. Para que este serviço seja executado com sucesso o usuário submetendo a aplicação deve ter realizado uma alocação de recursos.

Cancelamento de aplicação. Este serviço permite que um usuário cancele uma aplicação por ele submetida, identificada através da identificação fornecida no momento da execução da aplicação.

Monitoração de aplicações. Através deste serviço usuários podem acompanhar a execução de suas aplicações.

3.5.3 Gerência de arquivos

Este grupo contém serviços relacionados à transferência, gerência e armazenamento de arquivos de clientes.

Armazenamento temporário de arquivo. Este serviço permite que clientes submetam um ou mais arquivos que serão armazenados no *site* até que o cliente encerre a sua seção, o que pode ocorrer explicitamente (através do serviço de liberação) ou implicitamente (quando a capacidade computacional disponibilizada ao usuário for excedida).

Armazenamento persistente de arquivos. Este serviço habilita clientes a armazenarem arquivos permanentemente no *site* gerenciado pelo SRS. Isto pode ser útil para reduzir o tempo gasto com transferências de arquivos, porém exige a aplicação de uma política de cotas para evitar abuso por parte dos clientes.

Listagem e remoção de arquivos. Estes serviços permitem a realização de tarefas de gerência sobre arquivos armazenados no *site* através do SRS.

Download *de arquivos.* Através deste serviço usuários podem receber os arquivos gerados pelas suas aplicações, ou previamente armazenados.

3.5.4 Segurança e controle de acesso

Este grupo contém serviços relacionados a segurança e controle de acesso ao *site*. A aplicação destes serviços requer a adoção de mecanismos de segurança que forneçam autenticação e, se desejado por administradores e/ou clientes, encriptação de dados.

Identificação de cliente. Este serviço permite a identificação de clientes que solicitam serviços do SRS. Isto pode ser feito com a utilização de certificados X.509 [58], que é o formato de certificado mais utilizado em comércio eletrônico [59].

Controle de acesso. Este serviço permite que administradores do *site* restrinjam o acesso de determinados usuários ou grupos de usuários aos recursos locais. Isto é feito com a especificação, por parte do administrador do *site*, dos acessos a que os usuários ou (grupos) têm direito.

3.5.5 Economia

Serviços pertencentes a esta categoria são aqueles relacionados à cobrança dos clientes pela utilização dos recursos do *site*. Nesta categoria, dois serviços são propostos:

Cobrança por uso de recursos. Este serviço é invocado quando clientes precisam pagar pelo uso dos recursos (este pagamento pode se dar através de créditos reais, virtuais ou através de favores).

Saldo atual. Fornecimento do saldo atual de um determinado cliente.

3.6 Comparação entre os escalonamentos do Globus e SRS

Centros de pesquisa que utilizam *clusters* comumente o fazem objetivando aumentar o desempenho das aplicações. Neste cenário – de computação de alto desempenho – o objetivo é adquirir uma grande quantidade de poder computacional por pequenas quantidades de tempo. A utilização dos recursos é coordenada por um escalonador de *clusters* (CRM – *Cluster Resource Manager*). O CRM garante a um usuário acesso (geralmente exclusivo) a uma partição dos recursos do *cluster* por um tempo determinado.

Como diferentes CRM's podem ter formas de acesso diferentes, cabe ao GRAM prover a usuários da grade acesso uniforme a estes recursos. Ainda, como o acesso aos recursos é exclusivo por um tempo, o GRAM não precisa controlar possível preempção nas aplicações dos usuários.

O SRS, por sua vez, admite que recursos de um *cluster* sejam acessados por usuários da grade de forma oportunística, isto é, disponibilizados a usuários da grade quando não houverem usuários locais utilizando estes recursos. Então, uma diferença entre o SRS e o GRAM é que o primeiro deve ser capaz de tratar a preempção de recursos de usuários da grade, enquanto que o segundo não considera esta possibilidade.

A exceção ao discutido acima é quando o GRAM é utilizado para prover acesso a recursos controlados pelo Condor [43] . O Condor é um sistema que é utilizado para aproveitamento de ciclos ociosos em estações de trabalho e que também pode ser utilizado para gerenciar *clusters*. Neste caso, o próprio Condor gerencia a preempção de recursos e o escalonamento de aplicações de usuários.

A justificativa para que seja adotado o *Site Resource Scheduler* ao invés de GRAM e Condor é que o último não é adequado para *sites* que realizam computação de alto desempenho. O Condor é um sistema para computação de alta vazão, cujo objetivo é prover uma grande quantidade de processamento ao longo de uma grande quantidade de tempo [60]. Para *sites* que trabalham com alto desempenho, devem ser preferidos CRM's convencionais, otimizados para este fim. Neste caso, para permitir que usuários locais de recursos tenham acesso dedicado e os usuários da grade acesso não-dedicado aos recursos do *cluster*, o SRS deve ser adotado. Ainda, enquanto deve existir um GRAM para cada *cluster* de um *site*, um único SRS interage com todos os recursos do *site*.

O CSF é um *framework* para desenvolver *metaschedulers* utilizando componentes Globus. Ele recebe todas as requisições de usuários (assim como o SRS) mas confia nos serviços do GRAM para realizar alocações para as requisições. Logo, o CSF possui as mesmas limitações que o Globus para a utilização de recursos.

Outra diferença é que, enquanto o SRS gerencia os recursos de um *site*, o CSF controla o acesso a recursos de um conjunto de *sites*, sendo que alguns deles podem também ser acessados também por outros *metaschedulers*, ou seja, requisições por recursos podem chegar através de cada um dos *metaschedulers*, podendo existir uma situação de disputa por recursos entre *metaschedulers* que servem a organizações virtuais diferentes. O SRS, por sua vez, recebe todas as requisições da grade, e portanto pode controlar a distribuição de recursos entre os usuários.

A Figura 11 ilustra a diferença entre a utilização do CSF e do SRS. Em 11(a) é apresentado um *site* que utiliza o CSF. Diferentes recursos do *site* podem ser acessados por organizações virtuais diferentes. Cada uma utiliza um CSF para acessar estes recursos, e alguns deles podem ser acessados por ambos os CSF's (na figura, isso acontece com o Recurso 1 do *site* B). Em 11(b), o SRS é utilizado. Todos os recursos do *site* são acessados via SRS e os clientes requisitam acesso a recursos através dele, sendo o acesso liberado de acordo com os direitos de acesso do usuário. É importante notar que nesta figura, o cliente pode ser um usuário acessando diretamente, um *grid scheduler* e até mesmo um CSF, representando alguma organização virtual.

3.7 Comparação entre os escalonamentos do OurGrid e SRS

No modelo de grade do OurGrid, o usuário da grade gerencia os recursos que recebe, ou seja, ele submete aplicações e realiza as operações de transferência de arquivos. Se um dos recursos se torna indisponível, o próprio escalonador do usuário (o MyGrid) deve escalonar

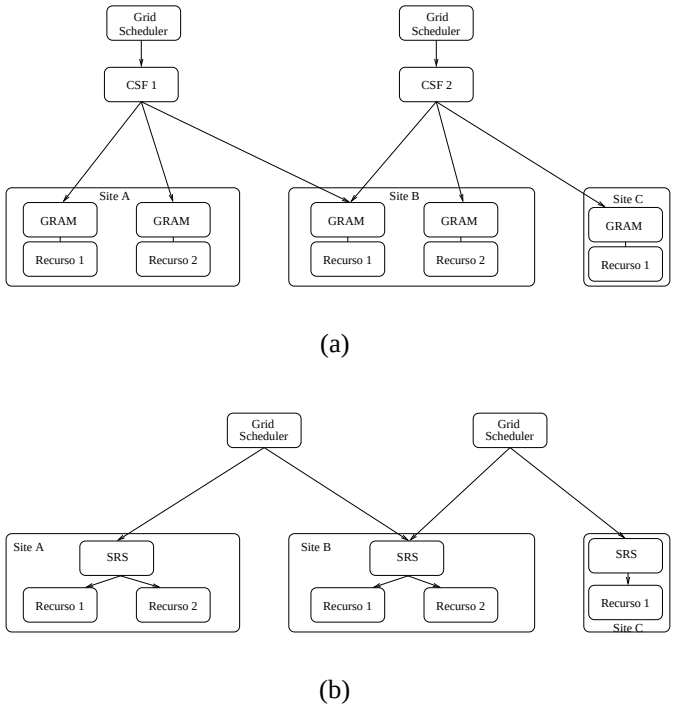

Figura 11 – Acesso a recursos via (a) CSF (b) SRS.

novamente a tarefa interrompida, realizando antes toda a transferência de arquivos previamente realizada. Eventualmente, esta transferência é para um outro recurso localizado no mesmo *site*.

Isto acarreta uma ineficiência que será mais significativa quanto mais voláteis foram os recursos. Em um *site* onde usuários locais sempre terão preferência sobre usuários da grade, interrupções na execução ocorrerão rotineiramente.

O SRS em um cenário como este reduz a transferência de arquivos entre o usuário e o *site*. Além do mais, o SRS pode reduzir ainda mais a troca de mensagens de controle entre clientes e grade, pois enquanto no OurGrid as tarefas são enviadas uma a uma para cada recurso, o SRS permite que as tarefas sejam enviadas em uma única etapa. Além disso, o SRS também retira o controle dos recursos do usuário e faz com que a quantidade de recursos a serem gerenciados seja reduzido, pois o usuário só acessa um recurso por *site*, e não os recursos reais que estão sendo efetivamente utilizados.

Quanto ao *peer* do OurGrid, este realiza a conexão com outros *peers* e descobre recursos para os usuários a eles conectados. Todavia, ele não realiza escalonamento, apenas controla a distribuição de recursos entre diferentes *peers* e usuários locais. O SRS por sua vez fornece recursos virtuais a usuários e ainda realiza o escalonamento das tarefas e gerência dos recursos do *site*.

3.8 Considerações finais sobre o SRS

Neste capítulo foi apresentada uma proposta de arquitetura para o *Site Resource Scheduler*. Foram considerados diversos componentes, cada um gerenciando um diferente aspecto que necessita estar presente em um *middleware* de grade. No entanto, a efetiva implantação do modelo necessita considerar também a presença de um *grid scheduler* capaz de explorar os recursos da mesma forma que o SRS. Alternativamente, pode ser adotado um modelo de abstração mais flexível e ser utilizado algum *grid scheduler* existente.

Ao invés do *Site Resource Scheduler* ser implementado na forma de um *middleware*, sua arquitetura modular pode ser aproveitada de forma que apenas alguns módulos selecionados por administradores de *sites* sejam utilizados em conjunto com um *middleware* existente. Neste caso, aqueles serviços realizados pelo sistema original podem ser aproveitados e apenas são acrescentados módulos do SRS que agreguem funcionalidades inexistentes no sistema. O estudo de caso apresentado no Capítulo 4 adota esta abordagem para o *middleware* OurGrid, e o Capítulo 5 descreve como esta abordagem pode ser utilizada no Globus.

4 Estudo de caso

Foi desenvolvido como estudo de caso para o *Site Resource Scheduler* um módulo para o OurGrid que implementa alguns dos seus serviços. Durante a concepção deste módulo, buscou-se obedecer a dois requisitos. O primeiro é que o protocolo de comunicação entre MyGrid e o Peer ao qual o cliente está associado, e o protocolo de comunicação entre *peers* não deve ser modificado. O segundo é que o código do OurGrid não deve ser alterado.

A seguir o desenvolvimento deste módulo será descrito com mais detalhes.

4.1 Implementação

A solução encontrada para a implementação do SRS obedecendo-se aos princípios previamente citados foi o seu desenvolvimento como um pacote Java com um *script* para disparar o programa com os parâmetros adequados. O módulo desenvolvido utiliza os pacotes do OurGrid para o aproveitamento das classes existentes.

Para respeitar o requisito de utilização dos protocolos existentes, foi necessário abrir mão de um modelo de definição de capacidade computacional mais sofisticado em favor da única unidade com a qual o MyGrid é capaz de negociar: máquinas.

O *broker* MyGrid requisita ao Peer um determinado número de máquinas (chamadas de GuMs no OurGrid). O número de máquinas requisitado é igual ao produto entre o número de tarefas a serem executadas (que é função da aplicação) e o número de réplicas geradas para cada uma (definido pelo usuário).

O SRS por sua vez disponibiliza um conjunto de máquinas virtuais (SRSGuMs) aos clientes. Este número é igual ao número de processadores (e não o de computadores) disponíveis no *site*. O número máximo de SRSGuMs criados pode ser definido pelo administrador. Uma GuM é descrita em função das propriedades da máquina que representa (no OurGrid, existe uma relação de 1:1 entre GuMs e máquinas reais). As SRSGuMS são descritas com as características da máquina que "ativou" a SRSGuM. Por exemplo, se uma máquina possui dois processadores Itanium 2 e 256MB de memória RAM, no momento que esta máquina é inserida na grade o SRS gerará duas SRSGuMs de arquitetura Itanium 2, cada uma com 256MB de RAM (e possuindo também as demais características presentes na máquina original).

Estas máquinas virtuais são criadas somente para dar uma estimativa da capacidade computacional do *site*: em algum determinado momento é possível que uma SRSGuM não esteja

mapeando nenhuma máquina do *site* (isto é, requisições enviadas à máquina são encaminhadas para a fila do escalonador).

SRSGuMs são então disponibilizadas aos usuários quando estes solicitam recursos. Porém, se no OurGrid uma GuM mapeia diretamente um recurso, e cada requisição de serviço é direcionada diretamente à máquina (especificamente, para o servidor *User Agent* da máquina) para que a aplicação seja executada, no SRS estes pedidos são encaminhados ao agente encarregado do processamento do serviço: `SRSScheduler`, no caso de execução de tarefas e `SRSFileProxy` no caso de manipulação de arquivos. Este direcionamento é feito no próprio SRS, e o cliente não está ciente de que ele não está acessando diretamente um recurso. Isto é possível porque o `SRSGateway` (o objeto responsável por direcionar as requisições) é um objeto remoto que implementa a interface `UserAgentServer`. Portanto, do ponto de vista do escalonador do usuário, a interação está ocorrendo como de costume, com um `UserAgentServer` em uma máquina remota.

4.1.1 Comparação entre o protótipo e o modelo geral do SRS

O protótipo desenvolvido não implementa todos os módulos apresentados na Figura 7. O mecanismo de contabilidade do uso de recursos utilizado é o original do OurGrid, e substitui o módulo de economia. O módulo de informações foi implementado e interage com as classes do OurGrid que monitoram o estado das máquinas da grade (estas substituem o módulo de gerência de recursos), a fim de fomentar o escalonador com estas informações. Os módulos de escalonamento e gerência de arquivos foram implementados, embora as operações relacionadas a armazenamento persistente de arquivos não estejam presentes nesta versão. O módulo de segurança não é implementado, e é o único módulo ausente no protótipo que não é substituído por uma funcionalidade equivalente do OurGrid: nenhuma consideração a respeito de segurança foi aplicada durante este trabalho.

4.2 Comunicação

A comunicação entre dois ou mais *peers*, entre o MyGrid e um Peer e entre o MyGrid e as GuMs ocorre via RMI. No OurGrid, cada GuM executa um RMI Registry no qual o `UserAgentServer` se registra e ao qual o MyGrid consulta para obter a referência aos objetos remotos. Com o SRS, os SRSGateways se registram em um Registry na mesma máquina que executa o SRS, e é este objeto que é consultado pelo MyGrid para obtenção das referências remotas.

Logo, no Registry da máquina que executa o SRS ficam registrados todos os objetos remotos criados pelo OurGrid mais um `SRSGateway` para cada SRSGuM criada no *site*. Cada

GuM e cada SRSGuM é um `OurGridMachine`. A diferença entre as duas é que na primeira o `Connector` acessa um `UserAgentServer` que, mesmo sendo acessado indiretamente (através de *Gateways*), termina acessando um `UserAgentServer` que executa diretamente no recurso da grade, enquanto que a SRSGuM não acessa diretamente o recurso, e sim o `SRSGateway`, conforme descrito anteriormente. Esta diferença está esquematizada na Figura 12: em 12(a), está o esquema de acesso atual do OurGrid, enquanto em 12(b) está representado o acesso realizado pelo SRS.

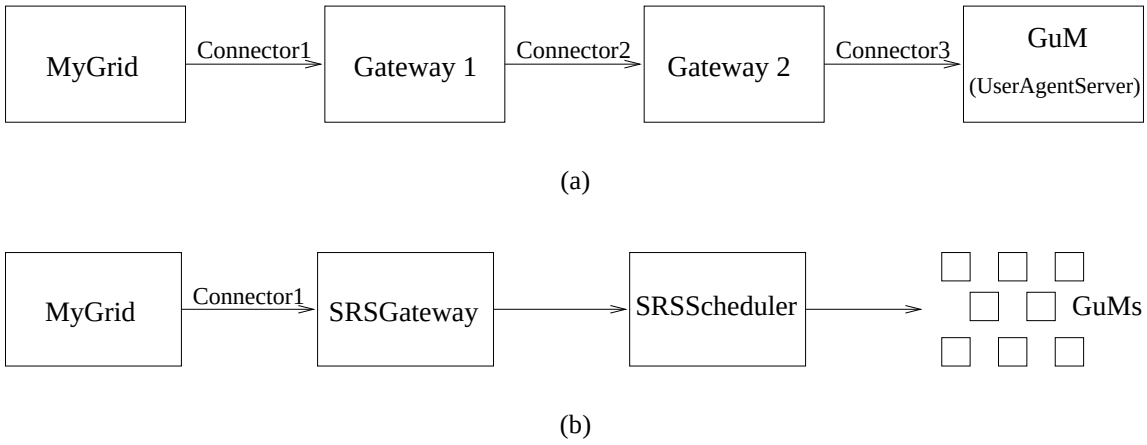

Figura 12 – Esquema de acesso aos recursos realizado pelo (a) OurGrid (b) SRS.

Um objeto da classe `SRSResourceManager` tem como função informar mudanças no estado das máquinas conhecidas por ele ao `SRSScheduler`. Objetos desta classe recebem notificações do `Doctor` do OurGrid sobre mudanças nos estados das máquinas reais. Caso seja constatada a perda de uma máquina, a tarefa que executava na máquina volta para fila para ser executada em um momento oportuno.

4.3 Execução de tarefas

O protocolo de execução de tarefas do MyGrid é diferente daquele proposto para o SRS. O MyGrid não envia à GuM informações sobre a tarefa. As informações sobre que arquivos devem ser transferidos para a grade, o que deve ser executado e que arquivos devem ser recebidos após a execução fazem parte da descrição das tarefas dos usuários processadas na própria *home machine*. A submissão de arquivos, execução da tarefa e recebimento de resultados ocorre nesta ordem controlado por um escalonador na *home machine*, enquanto que no modelo proposto para o SRS as tarefas são escalonadas pelo SRS a partir da descrição da aplicação transmitida pelo cliente.

As etapas envolvidas no processo de submissão de tarefas de tarefas para execução no MyGrid são as seguintes:

- O MyGrid obtém recursos de provedores de recursos;
- O MyGrid divide a aplicação em tarefas e estas em réplicas, que são a unidade mínima de execução do MyGrid;
- Para cada réplica é feita a submissão de arquivos necessários (isto é chamado de “Fase Inicial”). Estes arquivos são submetidos para um diretório temporário na GuM que existirá até o fim da execução da réplica;
- É executada a tarefa propriamente dita (esta fase é chamada “Fase Remota”);
- Os arquivos de saída são solicitados e recebidos da GuM. Esta é a “Fase Final”;
- O diretório temporário é destruído.

As fases Inicial, Remota e Final ocorrem em seqüência e nesta ordem.

Devido ao fato da descrição da tarefa não ser enviada para a GuM, o SRS não pode escalonar diretamente as tarefas. É preciso esperar que as solicitações de cada uma das fases chegue para que a tarefa possa ser executada. Para otimizar o processo, o SRS faz o seguinte:

- Existe, na máquina que executa o SRS, um diretório temporário para cada SRSGuM (que pode ser facilmente identificado pelo seu `SRSGateway` correspondente). Durante a Fase Inicial, os arquivos são transferidos para este diretório;
- Durante a Fase Remota, todo o diretório temporário é transferido para a máquina que executará o processo e o comando a ser executado é processado. Esta linha de comando também é armazenada;
- Se a máquina onde a tarefa é executada for retirada da grade, a tarefa pode ser escalonada para outra máquina. Neste caso, o diretório temporário é transferido para a máquina nova e a linha de comando é executada (o cliente não é informado da ocorrência);
- Durante a Fase Final, o arquivo solicitado pelo cliente é buscado na máquina real, gravado no diretório temporário na máquina que executa o SRS e só então transmitido ao cliente. Desta forma, se a transferência falhar, a máquina que executou a tarefa não precisa ser consultada novamente;
- Quando o cliente solicita a liberação do diretório temporário, o diretório transferido para a máquina da grade é apagado. Somente então a máquina poderá executar outra tarefa. Este cuidado é necessário porque o SRS não tem como saber quantos e quais arquivos foram gerados pela aplicação, então até que o cliente libere o recurso existe a possibilidade de serem recebidos novos pedidos de transferência de arquivos.

Outra diferença entre o escalonador do MyGrid e o escalonador do SRS é que o primeiro é capaz de submeter apenas uma réplica para cada máquina, e não para cada processador. Portanto, caso o cliente receba uma máquina multiprocessada, esta será subutilizada. O SRS é capaz de submeter mais de uma tarefa para cada máquina, e o faz de forma a aproveitar melhor os recursos multiprocessados.

4.4 Escalonamento das tarefas

A escolha de um algoritmo de escalonamento de tarefas em grades a ser utilizado não é uma tarefa trivial. Conforme discutido na Seção 1.1, cada algoritmo de escalonamento costuma ser testado em cenários que beneficiam a sua aplicação. Braun et al. [23] apresenta uma comparação entre onze heurísticas de escalonamento, utilizando até mesmo algoritmos genéticos, que apresentaram bons resultados a um alto custo de tempo de processamento do escalonamento.

O fato é que a eficiência de um determinado algoritmo está relacionado diretamente a características da aplicação. Sob esta justificativa, o sistema Legion [29] oferece um escalonador que mapeia tarefas em recursos de forma aleatória e delega aos usuários a tarefa de implantar escalonadores mais poderosos para suas aplicações.

Seguindo uma alternativa semelhante, o protótipo do SRS para o OurGrid oferece um algoritmo de escalonamento simples, listado no Algoritmo 1. Este algoritmo mapeia as tarefas de usuários em recursos não-dedicados do *site*.

Algoritmo 1: Escalonamento implementado no protótipo.

```
1 para cada Tarefa na fila de execução faça
2     para cada Máquina livre faça
3         se Máquina é compatível com SRSGuM então
4             Mapeia a tarefa a esta máquina;
5         fim se
6     fim para cada
7 fim para cada
```

Para cada uma das tarefas presentes na fila de execução é procurado um recurso compatível com a tarefa (Linha 4). A necessidade desta compatibilidade é resultante da forma como o OurGrid gerencia os recursos. Uma vez que a informação sobre os requisitos da tarefa em particular não chega no SRS, deve-se aproveitar o *matching* realizado pelo escalonador do usuário: quando este mapeou uma tarefa a um recurso, é porque este recurso cumpre os requisitos da tarefa. Como cada SRSGuM é criada com especificações baseadas em uma máquina real, o SRS procura recursos com especificações mínimas capazes de atender a tarefa. O Algoritmo 2 apresenta a forma como este *matching* é realizado.

Se o sistema operacional (Linha 1), tipo de acesso, isto é, se o acesso a máquina é feito pelo

Algoritmo 2: Matching implementado no protótipo.

```
   Entrada: E1: Especificação da SRSGuM
   Entrada: E2: Especificação da máquina real
 1 se E1.SistemaOperacional = E2.SistemaOperacional então
 2     se E1.TipoDeAcesso = E2.TipoDeAcesso então
 3         se E1.Arquitetura = E2.Arquitetura então
 4             se E1.Memória <= E2.Memória então
 5                 retorna Verdadeiro
 6             fim se
 7         fim se
 8     fim se
 9 fim se
10 retorna Falso
```

mesmo tipo de User Agent (Linha 2) e arquitetura (Linha 3) são os mesmos e a máquina real tem tanta ou mais memória que a quantidade especificada na SRSGuM (Linha 4), a SRSGuM e a máquina real são considerados compatíveis (Linha 5). Caso contrário, são considerados não compatíveis (Linha 10).

Um algoritmo mais poderoso pode ser implementado e incluído no módulo através do mecanismo de herança do Java.

4.5 Testes

Com o objetivo de analisar o efeito da gerência realizada pelos sistemas OurGrid e MyGrid, com e sem o SRS, no tempo de execução de aplicações, foram realizados alguns experimentos. Para evitar a inclusão de um protótipo que não implementa todos os serviços utilizados por clientes da grade em um sistema em produção (a comunidade OurGrid), foi montada uma comunidade específica para os testes, reproduzida na Figura 13. Neste ambiente, o Corepeer (serviço centralizado que registra os participantes da grade) e um Peer estavam situados no CPAD[1], em Porto Alegre, Rio Grande do Sul e um outro Peer e a *home machine* situados na Universidade Federal de Campina Grande (UFCG), na Paraíba.

O Peer CPAD possuía as máquinas do *cluster* Ombrófila, composto de 14 máquinas Pentium 3 de 1GHz e 14 máquinas Pentium 4 de 1.4GHz, todas monoprocessadas e com 256MB de RAM. O Peer da UFCG possuía apenas a *home machine*.

Para retirar dos testes a heterogeneidade do ambiente, facilitando a reprodução dos experimentos e eliminando um dos parâmetros dos testes, optou-se por uma aplicação que transfere um arquivo, espera uma determinada quantidade de tempo e transfere o arquivo de volta. A seguir, serão descritos os parâmetros utilizados nos testes e o motivo pelo qual foram escolhidos.

[1] Centro de Pesquisas em Alto Desempenho – PUCRS/HPB.

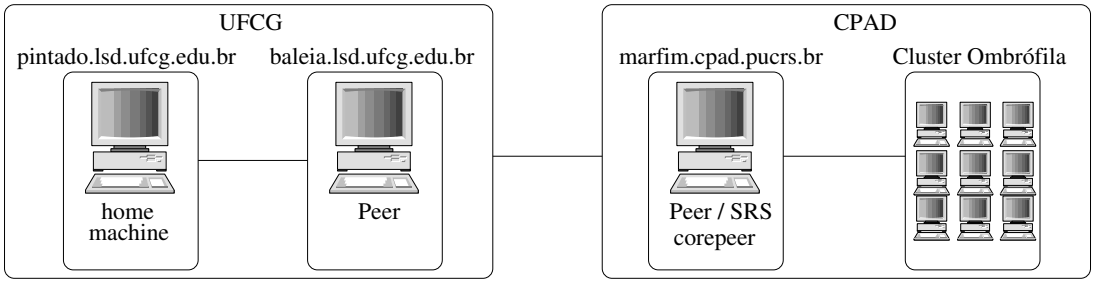

Figura 13 – Ambiente utilizado nos testes do SRS.

Tamanho dos arquivos transferidos. Foram escolhidos três tamanhos diferentes de arquivos, baseados em três aplicações reais que utilizam o ambiente OurGrid: arquivos de 100kB (tamanho de fotos que são processadas pelo MyPhotoGrid, aplicação que fazia parte das primeiras versões do MyGrid), arquivos de 1MB (tamanho médio de arquivos utilizados no GerPav-Grid[2], aplicação que objetiva gerenciar o estado dos pavimentos de Porto Alegre) e arquivos de 10MB (tamanho de arquivos encontrados no HPC-ICTM [61], que processa imagens de satélites). Também foi realizado um teste sem transferência de arquivos, com o intuito de analisar o *overhead* gerado pelo SRS.

Número de tarefas. Foi escolhido um número de tarefas igual a 7 para compor cada aplicação. Foi escolhido este número porque corresponde a 25% do número de máquinas disponíveis no *cluster* Ombrófila. Com isto foi possível garantir uma quantidade de recursos suficientes para executar as tarefas simultaneamente mesmo com algumas máquinas sendo retiradas deliberadamente da grade. Neste caso, novas máquinas eram disponibilizadas.

Forma de alocação do *cluster*. Durante os testes, os recursos do *cluster* foram utilizados de forma não-dedicada. Portanto, periodicamente alguma máquina podia ser retirada da grade e disponibilizada a usuários locais. Isto foi simulado com a remoção de uma máquina da grade a cada 10 minutos, sendo devolvida à mesma 10 minutos depois. O número de máquinas removidas da grade foi mantido o mesmo, para os testes com e sem o SRS.

Duração das tarefas. Foram escolhidas tarefas de 5 minutos, de forma que o tempo de execução da aplicação seja menor do que o tempo no qual uma máquina é removida.

Estes parâmetros foram então combinados e, para cada conjunto de parâmetros, dentre aqueles que manipulam arquivos menores de 10MB, foram realizadas 10 execuções da aplicação. O tempo total de execução da aplicação (tempo decorrido desde a submissão da aplicação à grade até o término da última tarefa que a compõe) mais alto e mais baixo foram descartados e uma média dos valores restantes foi feita. Para os testes que enviam e recebem arquivos de 10MB, foram realizadas apenas 5 simulações, e para o cálculo da média todos os valores foram considerados. Este procedimento foi realizado para o Peer CPAD com e sem o SRS.

[2] `http://www.cpad.pucrs.br/gerpavgrid/index.jsp`

O MyGrid foi configurado para não realizar replicação de tarefas. Portanto, cada vez que uma aplicação era submetida, era enviado ao Peer uma solicitação por uma quantidade de máquinas igual ao número de tarefas que compõe a aplicação. Também, durante os testes, sempre se garantiu que haveriam máquinas suficientes para atender todas as tarefas, pois o objetivo dos testes não é avaliar a interferência da carga do *cluster* na execução das tarefas. Um estudo sobre este assunto pode ser encontrado em [57].

4.6 Resultados

A Tabela 1 mostra os resultados obtidos nos testes. A coluna central exibe os tempo de execução obtidos em uma grade que utiliza o OurGrid sem o SRS, e a 3ª coluna exibe os resultados obtidos com o SRS. Em cada uma das linhas estão os resultados para um determinado tamanho de arquivo ou para a execução sem transferência de arquivos.

Tabela 1 – Resultados obtidos para 7 tarefas de 5 minutos.

Tamanho do arquivo	OurGrid	OurGrid+SRS
–	302s	303s
100KB	327s	330s
1MB	463s	474s
10MB	3212s	2055s

A inclusão do SRS incluiu um *overhead* no OurGrid que causou um aumento no tempo de execução da aplicação quando as aplicações envolviam a transferência de arquivos pequenos. Este *overhead* é causado pela necessidade de uma transferência local de arquivos (primeiro um arquivo é transferido ao SRS e depois para o recurso que o utilizará) e pela existência de um escalonador local, não presente no OurGrid.

Quando a aplicação exige a transferência de um volume maior de dados, o fato do arquivo ser primeiro armazenado pelo SRS e somente ser transferido ao recurso no momento da execução mostrou-se eficaz para reduzir o tempo de execução da aplicação. Quando não existe transferência de arquivos envolvida na execução da tarefa, o tempo de execução da aplicação das duas abordagens é equivalente, conforme mostrado na primeira linha da tabela.

Para verificar se o *overhead* introduzido pelo SRS é influenciado pelo número de tarefas que compõem a aplicação, foi realizado um teste que consistiu na execução de aplicações compostas por 1, 4, 7 e 10 tarefas de duração de 5 minutos e com transferência de arquivos de 1MB, tanto de entrada quanto de saída utilizando-se o mesmo ambiente do experimento anterior. Os resultados, mostrados na Tabela 2 mostram uma flutuação no tempo de execução, mas não proporcional ao número de tarefas, provavelmente causada por variação no tráfego pela rede ente Porto alegre e Campina Grande. O gráfico da Figura 14 permite uma melhor visualização da variação do tempo de execução.

Tabela 2 – Execução de aplicações no SRS com transferência de arquivos de 1MB.

Número de tarefas	Tempo de execução (s)
1	466
4	490
7	474
10	505

O aumento no número de tarefas não causou em todos os casos aumento no tempo de execução das aplicações. Este resultado permite atribuir o *overhead* gerado pelo SRS à gerência e escalonamento de tarefas e à transferência local de arquivos, e não ao número de tarefas em submetidas.

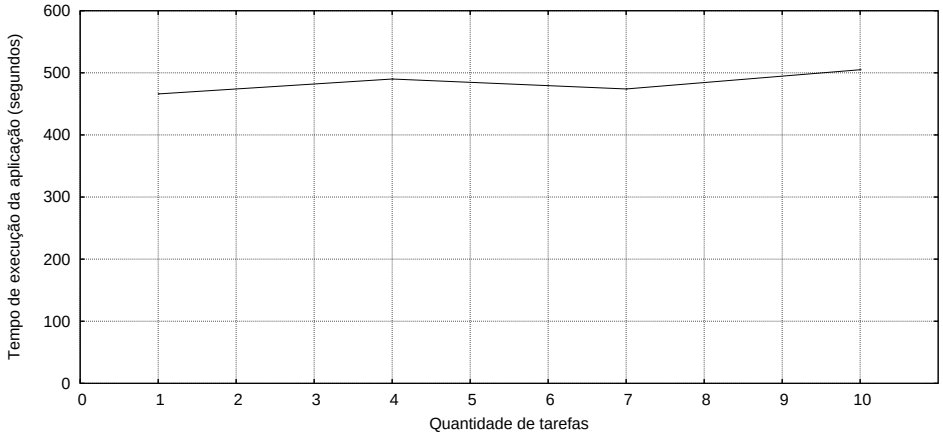

Figura 14 – Tempo de execução da aplicação por quantidade de tarefas (arquivos de 1MB).

Considerou-se também que, o algoritmo *Storage Affinity* do MyGrid, que procura utilizar recursos que já contenham os arquivos necessários pela tarefa, poderia ser uma abordagem mais eficiente no tratamento de transferências de arquivos do que o SRS. Realizou-se então o mesmo teste, com transferência de arquivos de 10MB utilizando-se este algoritmo de escalonamento. O resultado, mostrado na Tabela 3, mostra que, nas condições adotadas, este algoritmo não é superior ao WQR.

Tabela 3 – Execução de aplicações que transferem arquivos de 10MB.

Heurística de escalonamento	Tempo (s)
WorkQueue	3212
Storage Affinity	3178
WorkQueue+SRS	2055

De fato, o *Storage Affinity* é útil quando os arquivos já existem nos recursos. Um possível uso deste algoritmo é quando uma mesma aplicação será executada diversas vezes com arqui-

vos de entrada diferentes. Neste caso, os arquivos executáveis são armazenados e os dados são transferidos de forma convencional. Se estes arquivos de dados foram grandes, o ganho com a economia de transferência de dados não será grande. Porém o armazenamento de arquivos de dados grandes em recursos pode não ser possível, devido a limitações impostas por administradores, pois o diretório de armazenamento é compartilhado por todos os usuários da grade, e o espaço disponível neste diretório pode ser restringido por administradores.

Conclui-se que a inclusão do SRS em uma grade OurGrid pode diminuir o tempo de execução de tarefas que exigem grande volume de transferência de dados em *sites* que cedem à grade recursos de forma não-dedicada. Mesmo em *sites* que cedem recursos à grade de forma dedicada, ou para aplicações que transferem um pequeno volume de dados, o SRS pode ser utilizado com uma perda não significativa de desempenho das aplicações.

5 Sobre a implantação do SRS em uma grade Globus

O Globus Toolkit 4 é o padrão *de facto* em grades computacionais. Por isso, a implantação do SRS em uma grade Globus é importante para que o projeto alcance uma maior visibilidade. Neste capítulo será descrita a forma como o SRS pode ser implantado em um *site* que utiliza o *middleware*.

Antes de detalhar a processo de aplicação do SRS em Globus, é importante detalhar a arquitetura do GRAM (*Grid Resource* and *Allocation Management*), parte da arquitetura Globus responsável pela gerência de recursos e alocação em grades Globus.

5.1 GRAM

O GRAM é o componente do Globus responsável pelo gerenciamento dos recursos de um *site*. Ele deve estar presente em cada recurso (ou conjunto de recursos, no caso de *clusters* e MPP's) que poderá ser acessado através da grade. Embora já tenha sido descrito em linhas gerais no Capítulo 2, nesta seção este componente será apresentado de um ponto de vista arquitetural, ressaltando-se os seus componentes.

Clientes da grade acessam os recursos através do GRAM, por meio de uma API conhecida como RSL – *Resource Specification Language*.

RSL é uma linguagem de descrição que contém um conjunto de atributos que permitem que usuários descrevam os recursos necessários à aplicação a ser executada. Para solicitar recursos, usuários devem submeter ao GRAM um arquivo XML que respeita as especificações da RSL contendo a descrição dos recursos necessários às suas aplicações. Este arquivo é convertido pelo GRAM em requisições para o gerenciador de recursos local.

A Figura 15 (adaptada do manual do GRAM[1]) apresenta uma visão de alto nível dos componentes relacionados ao GRAM e de que forma eles se relacionam. Por questão de simplicidade, foram excluídos da figura todos os aspectos relacionados a credenciais de usuário.

O objetivo do GRAM é retirar do usuário e seu *grid scheduler* a tarefa de interagir com os possíveis escalonadores locais presentes em cada *site*: ao invés de conhecer e suportar os diferentes protocolos possíveis para alocação de recursos através destes escalonadores, o cliente só precisa conhecer um protocolo, o do GRAM, e este último realiza a operação no recurso. A conversão de operações enviadas para o GRAM (em RSL) para o comando do escalonador local

[1] `http://www.globus.org/toolkit/docs/4.0/execution/key/WS_GRAM_Approach.html`

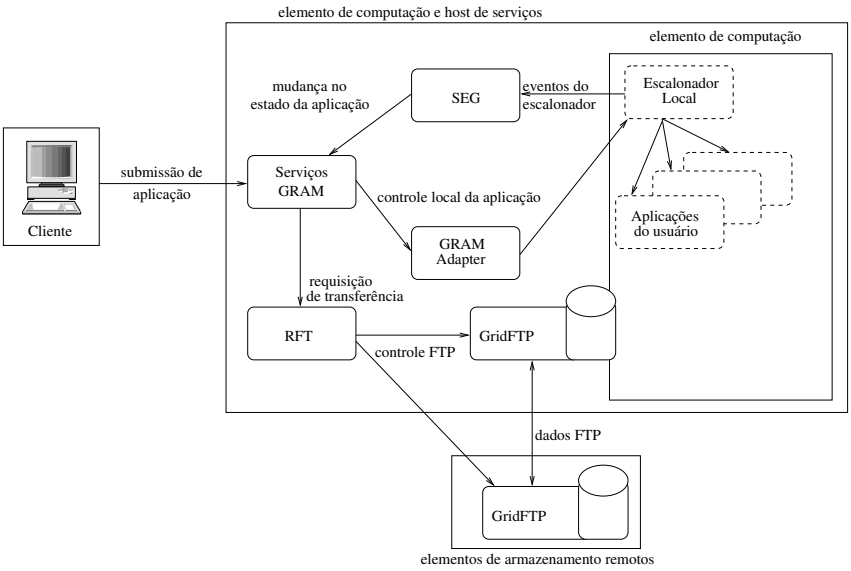

Figura 15 – Interação entre os diversos componentes do GRAM.

que realiza tal operação é realizado por um elemento do GRAM chamado *adapter*.

A distribuição Globus (pelo menos até a sua versão 4.0.1) disponibiliza *adapters* para o PBS (*Portable Batch System*), para o *software* proprietário LSF (*Load Sharing Facility*)[2], para o Condor [60] e para execução na mesma máquina que executa o GRAM (um *adapter* chamado de *fork*). Para acessar recursos controlados por outros gerenciadores, deve ser escrito um *adapter* específico, através de um *script* em linguagem Perl.

O SEG (*Scheduler Event Generator*) é o módulo do GRAM responsável por acompanhar o estado das aplicações. Este módulo comunica ao GRAM mudanças no estado das aplicações. Já as operações de envio e recebimento de arquivos são realizados pelo GridFTP. Como a transferência de arquivos pode ser longa, ela pode ser executada como um *grid service*, da mesma forma que tarefas em execução. Neste caso, a transferência é gerenciada por um outro componente do Globus, o RFT.

5.2 Adaptando o SRS ao Globus

Diversos centros de pesquisa realizam trabalhos relacionados ao Globus. Estas pesquisas consistem desde aplicações para as grades até *grid schedulers*. Para que o SRS seja aceito em um ambiente Globus, é importante que seja compatível com a maior parte (ou todos) estes projetos, e isto significa respeitar todos os protocolos existentes.

Duas formas possíveis de aplicar o SRS em uma grade Globus são:

- Através da disponibilização do SRS como um *grid service* autônomo, de forma compatível com o WS-*Resource Framework*;

[2] `http://www.platform.com`

- Através da inserção de um módulo dentro de uma máquina interagindo com os componentes do Globus, de uma forma análoga àquela aplicada no protótipo do OurGrid (Capítulo 4).

A solução apontada no item 1 não utiliza o GRAM: ela exige que o SRS implemente, além da gerência de recursos, a capacidade de transferência de arquivos e toda a gerência de credenciais. A solução 2 utiliza o GRAM e por isso pode aproveitar as funcionalidades relacionadas a segurança e transferência de arquivos. Por ser uma solução maias simples, ela será descrita em mais detalhes no decorrer deste capítulo.

5.2.1 Sobre a utilização do CSF

Deve ser feita uma consideração sobre a utilização do CSF para a implementação do SRS, aproveitando os serviços oferecidos por este. Pelo fato de o CSF funcionar em nível de organização virtual, e não em nível de *site*, somado ao fato de utilizar serviços do GRAM para gerenciar recursos locais, a alternativa 2 acima deveria ser empregada de qualquer forma, para permitir o acesso aos escalonadores de recursos de forma não-dedicada.

Portanto, como deverá ser realizado algum tipo de trabalho na interface entre o GRAM e o recurso para prover o acesso não-dedicado, confinar as modificações neste nível, e não adotar o CSF foi a estratégia escolhida para ser descrita.

5.3 SRS e GRAM

A Figura 16 posiciona o SRS no cenário apresentado na Figura 15. Em *sites* que possuem mais de um *cluster*, deve existir um GRAM para cada *cluster*. Com a proposta apresentada, passa a existir um único GRAM, pois do ponto de vista do último, existe apenas um "escalonador de recurso" abaixo dele (o SRS). É o SRS que conhece e gerencia todos os recursos do *site* (Figura 17).

Neste cenário (baseado em uma proposta geral para integração entre *clusters* e grades [57]), o SRS está localizado entre o GRAM adapter e o escalonador do cluster. Cabe ao SRS manter uma lista de recursos disponíveis à grade (informação que pode ser obtida através do gerenciador do cluster) e a lista de aplicações da grade (recebidas através do GRAM adapter) e providenciar o *matching* e execução das tarefas. O SRS também deve se comunicar com o SEG a fim de informar o estado da execução das tarefas.

Um GRAM adapter específico deve ser escrito, convertendo instruções RSL em instruções para o SRS. Todo o acesso aos recursos é feito via SRS, portanto o GRAM adapter não realizará nenhum acesso a eles.

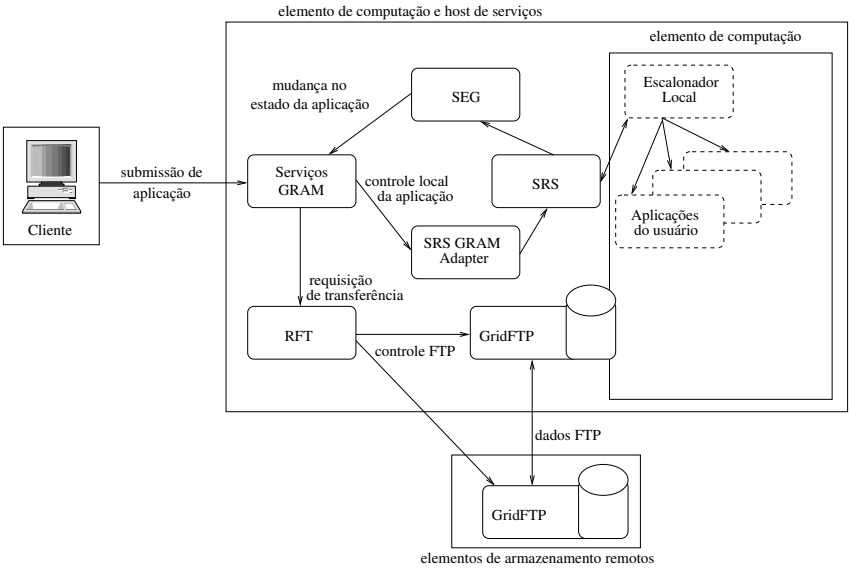

Figura 16 – Integração do SRS ao GRAM.

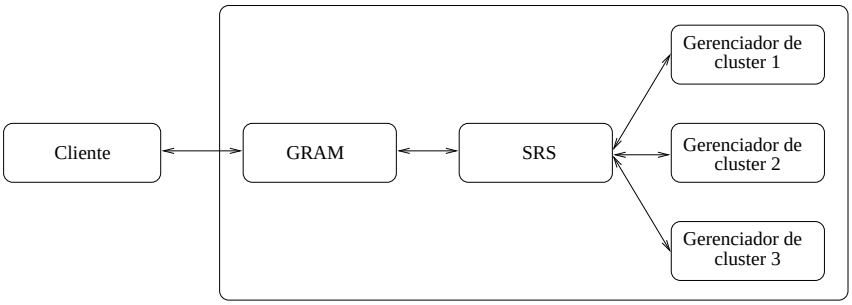

Figura 17 – SRS e GRAM em *site* com diversos *clusters*.

Os serviços relacionados à transferência de arquivos não precisam ser implementados, pois um servidor GridFTP presente no *host* do SRS possui a mesma função do módulo de gerência de arquivos: ele recebe os arquivos de servidores remotos e os disponibiliza aos recursos do *site*. Os serviços de segurança também não precisam ser implementados pois são realizados pelo Globus.

Existe um tipo de aplicação suportado pelo Globus que é de especial interesse para um site utilizando o SRS: os *multijobs*. Neste tipo de aplicação, são submetidos de uma única vez ao GRAM um conjunto de tarefas que devem ser executadas na grade. Este tipo de aplicação é interessante em um *site* gerenciado pelo SRS porque permite que o SRS controle todo o processo de submissão e execução das tarefas, algo que não acontece quando tarefas são submetidas seqüencialmente a máquinas virtuais.

5.4 Adicionando capacidade computacional abstrata

O RSL não possui entre os seus atributos um capaz de proceder a requisição de recursos na forma de unidade abstrata. Porém o RSL é extensível, permitindo que elementos de outros *namespaces* possam ser adicionados aos requisitos dos usuários. Quando estes elementos são encontrados em uma requisição de usuários, ele é propagado para o GRAM adapter. Para ser possível implantar esta funcionalidade, é necessária a criação de uma descrição em XML dos novos elementos oferecidos. Este arquivo deve ser publicado para que possa ser consultado por *parsers* XML. A posterior tradução da unidade abstrata em recursos reais deve ser tratada pelo SRS.

Os *grid schedulers* devem ser capazes de solicitar recursos utilizando esta unidade abstrata. Portanto, seria necessário o desenvolvimento de tal escalonador para o aproveitamento desta possibilidade. O GRAM não precisa ser modificado no processo: os novos atributos são passados ao GRAM adapter da mesma forma que os atributos pré-definidos.

5.5 Adicionando serviços sem estado ao SRS

Uma questão importante sobre serviços em sistemas distribuídos diz respeito a necessidade de um serviço armazenar ou não informações de estado, isto é, se o serviço possui uma "memória" a respeito de execuções passadas [62]. Citando um exemplo orientado a *web services*, um serviço de previsão de tempo pode ser implementado como um serviço sem estado (*stateless*): o resultado da previsão do tempo não é influenciado pelos acessos ao serviço até o momento. Um exemplo de *web services* com estado (*stateful*) é um serviço que retorna o número de acessos realizados até o momento. Cada vez que um usuário acessar o serviço, obterá um resultado diferente. O WS-*Resource Framework* é uma especificação que determina uma forma de oferecer *web services* com estados, pois serviços de grade armazenam informações de estado.

No que diz respeito a adaptação do SRS ao Globus, os serviços de transferência e armazenamento de arquivos, submissão e monitoração de aplicações e segurança são realizados pelo próprio Globus, que utiliza o WS-*Resource Framework* para empregá-los. A alocação de recursos e escalonamento de tarefas são oferecidos pelo SRS e também devem ser implementados através do WS-*Resource Framework*.

Entre os serviços oferecidos pelo SRS descritos no Capítulo 3, os serviços de fornecimento da capacidade momentânea do *site* e de fornecimento do saldo do usuário para utilização de recursos não são oferecidos nem pelo GRAM nem pelo SRS. Pelo fato destes serviços serem sem estado, eles podem ser oferecidos na forma de *web services* comuns, o que exige a definição da interface de serviço através de um arquivo WSDL, implementação e publicação do serviço [33]. Neste caso o *grid scheduler* também deve ser capaz de solicitar estes serviços e interpretar

o resultado de retorno.

5.6 Comparação entre a proposta e o modelo geral do SRS

Os módulos apresentados na descrição do SRS, Figura 7 estão presentes da seguinte forma nesta proposta:

- O módulo de segurança é substituído pelos serviços de segurança do Globus;
- O módulo de economia é implementado como um *web service* sem estado;
- O módulo de interface de usuário é substituído pelo GRAM;
- O módulo de informação é substituído pelo MDS;
- O módulo de gerência de arquivos é substituído pelo GridFTP;
- O módulo de escalonamento é implementado, recebendo tarefas do GRAM *Adapter* e distribuindo-as aos recursos apropriados;
- O módulo de gerência de recursos é implementado, recebendo informações sobre o estado dos recursos diretamente do escalonador do *cluster*.

5.7 Resumo da abordagem

A adaptação do SRS ao Globus é possível com a inclusão de um módulo entre o GRAM e o escalonador do *cluster* que mantém a lista de recursos disponíveis a grade e mapeia tarefas a elas. Tarefas da grade são remetidas ao SRS através de um GRAM *adapter* específico para o SRS. A seqüência de eventos durante a execução de uma aplicação nesta abordagem é a seguinte:

- O *grid scheduler* do usuário realiza uma consulta a um *web service* que retorna a capacidade momentânea do *site*. Alternativamente, este serviço pode retornar também o custo para a utilização destes recursos;
- Para a realização das etapas seguintes, é necessária a delegação de credenciais do usuário ao GRAM, para que este possa autenticar o usuário;
- Baseado na resposta retornada no primeiro passo, a aplicação (ou uma parte da aplicação) é submetida ao *site*, via GRAM e na forma de um *multijob*;

- No mesmo arquivo de descrição de tarefas que contém a especificação do executável também são transmitidos os comandos de transferência de arquivos, o que inclui o tipo de armazenamento que deve ser realizado (temporário ou persistente). O controle da transferência de arquivos é realizado pelo RFT utilizando o GridFTP;

- O *grid scheduler* se registra ao GRAM para receber notificações de eventos da sua aplicação;

- Após a execução, os arquivos são retornados. Novamente, o RFT é responsável pela coordenação da ação.

No esquema proposto para a adaptação do SRS ao Globus, o modo como ocorre a interação ente usuários e serviços não é alterado, apenas é incluída uma etapa adicional realizada antes da submissão da aplicação. As demais ações do SRS ocorrem dentro do *site* e são transparentes aos usuários e ao GRAM.

O SRS possa ser empregado no Globus sem exigir alteração nos programas dos clientes. Neste caso, são fornecidos aos clientes máquinas virtuais, da mesma forma que a empregada no protótipo do modelo. Porém uma exploração maior das possibilidades do *Site Resource Scheduler* só é possível com o desenvolvimento de um *grid scheduler* ciente da presença do *software*, ou seja, capaz de explorar a abstração oferecida pela abordagem.

6 Conclusão e trabalhos futuros

Cada vez mais centros de computação no mundo inteiro têm cedido recursos a grades. Enquanto a quantidade de recursos disponíveis nestas grades aumenta, a escalabilidade de *grid schedulers* torna-se um aspecto cada vez mais importante na área. Delegar a usuários a tarefa de gerenciar centenas (talvez milhares) de recursos espalhados pelo mundo e possuindo diferentes características leva a uma perda de desempenho na execução de tarefas, devido ao *overhead* inserido pela gerência e escalonamento de tarefas em tais recursos.

Neste trabalho foi apresentada uma nova estratégia de gerência e escalonamento de recursos em grades computacionais, chamada de *Site Resource Scheduler* (SRS). O SRS simplifica a visão que usuários têm dos recursos da grade e conseqüentemente a sua gerência, fazendo com que não mais acessem diretamente uma grande quantidade de recursos mas sim máquinas virtuais que representam a capacidade computacional disponível momentaneamente a eles em cada um dos *sites* aos quais têm acesso. Para que isto seja possível uma nova camada de abstração, responsável por virtualizar os recursos do *site*, é acrescentada à grade.

O acréscimo desta camada de abstração exige a definição de alguns características adicionais que não são necessárias nos sistemas convencionais, que são a definição de uma medida de capacidade computacional de recursos, a definição de uma forma de usuários expressarem as suas necessidades computacionais e a definição de uma maneira de garantir uma distribuição adequada dos recursos.

No decorrer do texto o tratamento a estas questões foi abordado, e uma arquitetura geral do SRS, com ênfase aos serviços de virtualização, foi apresentada. O modelo é genérico o bastante para ser implementado como um *middleware* próprio ou ser adaptado como um módulo de um *middleware* existente.

Foi desenvolvido também um protótipo do SRS para o *middleware* OurGrid, e um teste de desempenho, comparando-o com o OurGrid sem o SRS foi realizado. Concluiu-se que a inclusão do SRS em uma grade OurGrid pode diminuir o tempo de execução de tarefas que exigem grande volume de transferência de dados em *sites* que cedem à grade recursos de forma não-dedicada. Mesmo em *sites* que cedem recursos à grade de forma dedicada, ou para aplicações que transferem um pequeno volume de dados, o SRS pode ser utilizado com uma perda não significativa de desempenho das aplicações.

O trabalho foi concluído com uma proposta para a inclusão do SRS em grades Globus. A proposta não exige a modificações nos componentes do Globus, e é aplicada como um módulo de *software* entre o GRAM (componente de gerência de recursos do Globus) e o escalonador (ou escalonadores) de *clusters* do *site*.

Como possíveis trabalhos futuros derivados deste trabalho estão:

- Implementação completa do SRS no OurGrid;
- Implementação do SRS no Globus;
- Desenvolvimento de heurísticas de escalonamento;
- Aplicação de políticas de alocação de recursos;
- Otimização de utilização de recursos não-dedicados;
- Implementação de *Metascheduling*.

A inclusão de outras funcionalidades no SRS, além daquelas apresentadas neste trabalho poderá contribuir para um aproveitamento ainda melhor dos recursos computacionais de centros de computação, laboratórios de pesquisa, universidades, empresas e quem mais tiver interesse em todas as possibilidades oferecidas pelas grades computacionais.

Referências

[1] CZAJKOWSKI, K.; FOSTER, I.; KESSELMAN, C. Agreement-Based Resource Management. *Proceedings of the IEEE*, IEEE, v. 93, n. 3, p. 631–643, mar. 2005.

[2] SCHOPF, J. M. Ten Actions When Grid Scheduling. Em: NABRZYSKI, J.; SCHOPF, J. M.; WĘGLARZ, J. (Ed.). *Grid Resource Management: State of the Art and Future Trends*. Norwell: Kluwer Academic Publishers, 2003. cap. 2, p. 15–23.

[3] DE ROSE, C. A. F.; NAVAUX, P. O. A. *Arquiteturas Paralelas*. Porto Alegre: Editora Sagra Luzzatto, 2003. (Série Livros Didáticos).

[4] FOSTER, I.; KESSELMAN, C.; TUECKE, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *The International Journal of High Performance Computing Applications*, SAGE publications, v. 15, n. 3, p. 200–222, ago. 2001.

[5] BERMAN, F.; FOX, G.; HEY, T. The Grid: past, present, future. Em: BERMAN, F.; FOX, G. G.; HEY, A. J. G. (Ed.). *Grid Computing: Making the Global Infrastructure a Reality*. West Sussex: John Wiley and Sons, 2003. cap. 1, p. 9–50.

[6] CHETTY, M.; BUYYA, R. Weaving Computational Grids: How Analogous Are They with Electrical Grids? *Computing in Science and Engineering*, AIP and IEEE Computer Society, v. 4, n. 4, p. 61–71, jul. 2002.

[7] FOSTER, I. et al. The Phisiology of the Grid. Em: BERMAN, F.; FOX, G. G.; HEY, A. J. G. (Ed.). *Grid Computing: Making the Global Infrastructure a Reality*. West Sussex: John Wiley and Sons, 2003. cap. 8, p. 217–249.

[8] CIRNE, W. Grids Computacionais: Arquitetura, Tecnologias e Aplicações. Em: *Anais:* 3^a *Escola Regional de Alto Desempenho*. Santa Maria: Sociedade Brasileira de Computação, 2003. p. 103–134.

[9] NÉMETH, Z.; SUNDERAM, V. Characterizing Grids: Attributes, Definitions and Formalisms. *Journal of Grid Computing*, Springer, v. 1, n. 1, p. 9–23, mar. 2003.

[10] BAKER, M.; BUYYA, R.; LAFORENZA, D. Grids and Grid Technologies for Wide-Area Distributed Computing. *Software: Practice and Experience*, Wiley, v. 32, n. 15, p. 1437–1466, dez. 2002.

[11] WELCH, V. et al. Security for Grid Services. Em: *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*. Seattle: IEEE, 2003. p. 48–57.

[12] BUYYA, R. et al. Economic Models for Resource Management and Scheduling in Grid Computing. *Concurrency and Computation: Practice and Experience*, Wiley, v. 14, n. 13–15, p. 1507–1542, nov.,dez. 2002.

[13] MEDEIROS, R. et al. Faults in Grids: Why are they so bad and What can be done about it? Em: *Proceedings of the 4th International Workshop on Grid Computing*. Phoenix: IEEE Computer Society, 2003. p. 18–24.

[14] CASANOVA, H. et al. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. Em: *Proceedings of the 9th Heterogeneous Computing Workshop*. Cancun: IEEE Computer Society, 2000. p. 349 – 363.

[15] CIRNE, W. et al. Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach. Em: *Proceedings of the 2003 International Conference on Parallel Processing*. Kaohsiung: IEEE Computer Society, 2003. p. 407–416.

[16] COOPER, K. et al. New Grid Scheduling and Rescheduling Methods in the GrADS Project. Em: *Proceedings of the 18th International Parallel and Distributed Processing Symposium (CD-ROM)*. Santa Fe: IEEE Computer Society, 2004.

[17] GROPP, W. et al. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, Elsevier, v. 22, n. 6, p. 789–828, jan. 1996.

[18] CZAJKOWSKI, K. et al. Grid Service Level Agreement. Em: NABRZYSKI, J.; SCHOPF, J. M.; WĘGLARZ, J. (Ed.). *Grid Resource Management: State of the Art and Future Trends*. Norwell: Kluwer Academic Publishers, 2003. cap. 8, p. 119–134.

[19] BERMAN, F.; WOLSKI, R. Scheduling from the Perspective of Application. Em: *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing*. Syracuse: IEEE Computer Society, 1996. p. 100–111.

[20] VADHIYAR, S. S.; DONGARRA, J. J. A Metascheduler for The Grid. Em: *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*. Edinburgh: IEEE Computer Society, 2002. p. 343–351.

[21] HAMSCHER, V. et al. Evaluation of Job-Scheduling Strategies. Em: *Proceedings of the International Workshop on Grid Computing 2000*. Bangalore: IEEE Computer Society, 2000. p. 191–202.

[22] SUBRAMANI, V. et al. Distributed Job Scheduling on Computational Grids Using Multiple Simultaneous Requests. Em: *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*. Edinburgh: IEEE Computer Society, 2002. p. 359–366.

[23] BRAUN, T. D. et al. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, Elsevier, v. 61, n. 6, p. 810–837, jun. 2001.

[24] MAHESWARAN, M. et al. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. Em: *Proceedings of the 8th Heterogeneous Computing Workshop*. San Juan: IEEE Computer Society, 1999. p. 30–44.

[25] RANGANATHAN, K.; FOSTER, I. Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. *Journal of Grid Computing*, Springer, v. 1, n. 1, p. 53–62, mar. 2003.

[26] CATLETT, C. Standards for Grid Computing: Global Grid Forum. *Journal of Grid Computing*, Springer, v. 1, n. 1, p. 3–7, mar. 2003.

[27] LESYNG, B.; BALA, P.; ERWIN, D. EUROGRID – European computational grid testbed. *Journal of Parallel and Distributed Computing*, Elsevier, v. 63, n. 5, p. 503–610, maio 2003.

[28] CHIEN, A. A.; MARLIN, S.; ELBERT, S. T. Resource Management in the Entropia System. Em: NABRZYSKI, J.; SCHOPF, J. M.; WĘGLARZ, J. (Ed.). *Grid Resource Management: State of the Art and Future Trends*. Norwell: Kluwer Academic Publishers, 2003. cap. 26, p. 431–450.

[29] GRIMSHAW, A. S.; WULF, W. A. Legion– A View From 50,000 Feet. Em: *Proceedings of the 5th IEEE International High Performance Distributed Computing, 1996*. Syracuse: IEEE Computer Society, 1996. p. 89–99.

[30] GRIMSHAW, A. S. et al. From Legion to Avaki: the persistence of vision. Em: BERMAN, F.; FOX, G. G.; HEY, A. J. G. (Ed.). *Grid Computing: Making the Global Infrastructure a Reality*. West Sussex: John Wiley and Sons, 2003. cap. 10, p. 265–298.

[31] FOSTER, I. Globus Toolkit Version 4: Software for Service-Oriented Systems. Em: *IFIP International Conference on Network & Parallel Computing*. Beijing: Springer-Verlag, 2005. (Lecture Notes in Computer Science, v. 3779), p. 2–13.

[32] FOSTER, I.; KASSELMAN, C. The Globus Project: A Status Report. Em: *Proceedings of the Seventh Heterogeneous Computing Workshop*. Orlando: IEEE Computer Society, 1998. p. 4–18.

[33] ALONSO, G. et al. *Web Services: Concepts, Architectures and Applications*. Berlin: Springer, 2004.

[34] FOSTER, I. et al. Modeling and Managing State in Distributed Systems: The Role of OGSI and WSRF. *Proceedings of the IEEE*, IEEE, v. 93, n. 3, p. 604–612, mar. 2005.

[35] Internet Engineering Task Force. *The TLS Protocol version 1.0*. jan. 1999. RFC 2246. Disponível em http://www.ietf.org/rfc/rfc2246.txt. Último acesso em 18 de novembro de 2005.

[36] GOLDFARB, C. F.; PRESCOD, P. *The XML handbook*. 3^a. Upper Saddle River: Prentice Hall, 2001.

[37] FOSTER, I. *A Globus Toolkit Primer*. 2006. Disponível em http://www-unix.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf. Último acesso em 25 de janeiro de 2006.

[38] Platform Computing. *Open source metascheduling for Virtual Organizations with the Community Scheduler Framework (CSF)*. 2003. Technical Whitepaper. Disponível em http://www2.platform.com/resources/whitepapers. Último acesso em 18 de novembro de 2005.

[39] ANDRADE, N. et al. OurGrid: An approach to easily assemble grids with equitable resource sharing. Em: *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*. Seattle: Springer-Verlag, 2003. p. 61–86.

[40] CIRNE, W. et al. Scheduling in Bag-of-Task Grids: The PAUÁ Case. Em: *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing*. Foz do Iguaçu: IEEE Computer Society, 2004. p. 124–131.

[41] IN, J. et al. Policy Based Scheduling for Simple Quality of Service in Grid Computing. Em: *Proceedings of the 18th International Parallel and Distributed Processing Symposium (CD-ROM)*. Gainesville: IEEE Computer Society, 2004.

[42] IN, J. et al. SPHINX: A Fault-Tolerant System for Scheduling in Dynamic Grid Environments. Em: *Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium (CD-ROM)*. Denver: IEEE Computer Society, 2005.

[43] LITZKOW, M. J.; LIVNY, M.; MUTKA, M. W. Condor – A Hunter of Idle Workstations. Em: *8th IEEE International Conference on Distributed Computing Systems*. San Jose: IEEE, 1988. p. 104–111.

[44] FREY, J. et al. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Journal of Cluster Computing*, Springer, v. 5, n. 3, p. 237–246, jul. 2002.

[45] BUYYA, R.; ABRAMSON, D.; GIDDY, J. Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. Em: *Proceedings of the Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region*. Beijing: IEEE Computer Society, 2000. v. 1, p. 283–289.

[46] ABRAMSON, D.; BUYYA, R.; GIDDY, J. A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. *Future Generation Computer Systems Journal*, Elsevier, v. 18, n. 8, p. 1061–1074, out. 2002.

[47] HUEDO, E.; MONTERO, R. S.; LLORENTE, I. M. Experiences on Adaptative Grid Scheduling of Parammeter Sweep Applications. Em: *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*. A Coruña: IEEE Computer Society, 2004. p. 28–33.

[48] HUEDO, E.; MONTERO, R. S.; LLORENTE, I. M. A framework for adaptive execution in grids. *Software: Practice and Experience*, Wiley, v. 34, n. 7, p. 631–651, mar. 2004.

[49] XIAO, L. et al. GridIS: an Incentive-based Grid Scheduling. Em: *Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium (CD-ROM)*. Denver: IEEE Computer Society, 2005.

[50] GEE, J. D. et al. Cache Performance of the SPEC92 Benchmark Suite. *IEEE Micro*, IEEE Computer Society, v. 13, n. 4, p. 17–27, ago. 1993.

[51] KOO, R.; TOUEG, S. Checkpointing and Rollback-Recovery for Distributed Systems. *IEEE Transactions on Software Engineering*, IEEE Computer Society, v. 13, n. 1, p. 23–31, jan. 1987.

[52] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *The unified modeling language user guide*. Reading: Addison-Wesley, 1999.

[53] BAKER, M.; BUYYA, R. Cluster Computing at a Glance. Em: BUYYA, R. (Ed.). *High Performance Cluster Computing*. Upper Sadle River: Prentice Hall, 1999. v. 1, cap. 1, p. 3–47.

[54] KELLER, A.; REINEFELD, A. CCS Resource Management in Networked HPC Systems. Em: *Proceedings of the Seventh Heterogeneous Computing Workshop*. Orlando: IEEE Computer Society, 1998. p. 44–56.

[55] JETTE, M.; GRONDONA, M. SLURM: Simple Linux Utility for Resource Management. Em: *The Fourth LCI International Conference on Linux Clusters: The HPC Revolution*. San Jose: LCI, 2003. Disponível em http://linuxclustersinstitute.org/Linux-HPC-Revolution/Archive/PDF03/Jette_M.pdf. Último acesso em 25 de janeiro de 2006.

[56] NETTO, M. A. S.; DE ROSE, C. A. CRONO: A configurable and Easy to Maintain Resource Manager Optimized for Small and Mid-Size GNU/Linux Cluster. Em: *Proceedings of the 2003 International Conference on Parallel Processing*. Kaohsiung: IEEE Computer Society, 2003. p. 555–562.

[57] NETTO, M. A. S. et al. Transparent Resource Allocation to Exploit Idle Cluster Nodes for Execution of Independent Grid Tasks. Em: *Proceedings of the First International Conference on e-Science and Grid Computing*. Melbourne: IEEE Computer Society, 2005. p. 238–245.

[58] ITU-T. *The directory: authentication framework*. 1997. Recommendation X.509. Disponível em http://www.itu.int/ITU-T/asn1/database/itu-t/x/x509/1997. Último acesso em 18 de novembro de 2005.

[59] NEUMAN, C. Security, Accounting, and Assurance. Em: FOSTER, I.; KESSELMAN, C. (Ed.). *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco: Morgan Kaufmann Publishers, 1998. cap. 16, p. 395–422.

[60] THAIN, D.; TANNENBAUM, T.; LIVNY, M. Condor and the Grid. Em: BERMAN, F.; FOX, G. C.; HEY, A. J. G. (Ed.). *Grid Computing: Making the Global Infrastructure a Reality*. West Sussex: John Wiley and Sons, 2003. cap. 11, p. 299–336.

[61] AGUIAR, M. et al. HPC-ICTM: the Interval Categorizer Tessellation-Based Model for High Performance Computing. Em: *Workshop on State-of-the-art in Scientific Computing (PARA)*. Lyngby: Springer-Verlag, 2005. p. 1–10.

[62] COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed Systems Concepts and Design*. 3^a ed. Edinburgh Gate: Addison-Wesley, 2001.

Livros Grátis

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

Milhares de Livros para Download:

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