InteGrade: Rumo a um Sistema de Computação em Grade para Aproveitamento de Recursos Ociosos em Máquinas Compartilhadas*

Andrei Goldchleger Fabio Kon Alfredo Goldman vel Lejbman
Marcelo Finger Siang Wun Song

Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo
{andgold,kon,gold,mfinger,song}@ime.usp.br

Setembro de 2002

Resumo

Computational Grids are a new trend in distributed computing systems. They allow the sharing of geographically disributed resources in an efficient way, extending the boundaries of what we perceive as distributed computing. Various sciences can benefit from the use of Grids to solve CPU-intensive problems, creating potencial benefits to the entire society. With further developement of Grid technology, it is very likely that corporations, universities, and public institutions will exploit Grids to enhance their computing infrastructure. In recent years, there has been a large increase in Grid technology research, which has produced some reference Grid implementations. This technical report presents the main features of four major Grid projects: Globus, Legion, Condor, and BOINC. We describe the design principles and the most important aspects of each system from our point of view. Finally, we introduce a new research project started recently at the University of São Paulo: *InteGrade*. In this new research effort, we aim at extending previous Grid research by using advanced distributed object technologies to build a novel Grid infrastructure to enable the use of idle processing time in commodity workstations without loss of Quality of Service for workstation users.

*A elaboração deste projeto foi financiada por recursos do CNPq (Kit Enxoval processos 68.0118/01-2 e 68.0075/01-1) e da FAPESP (processo 01/03861-0). Estamos no momento em busca de financiamento para a sua implementação.

1 Introdução

A computação atingiu um patamar surpreendente em termos da qualidade de recursos. Processadores que hoje equipam computadores pessoais possuem capacidade de processamento encontrada em supercomputadores há menos de uma década; as redes de computadores estão cada vez mais rápidas e disseminadas e o software evolui juntamente tornando-se cada vez mais modular, configurável e dinâmico.

Mesmo com os avanços em hardware, certos problemas ainda persistem. Alguns tipos específicos de usuários necessitam de grande poder computacional e não o possuem, ao passo que, em um instante qualquer, milhões de computadores pessoais estão ociosos ao redor do mundo. A Internet trouxe uma maneira inédita de conexão entre sistemas, mas certos recursos ainda não podem ser compartilhados e acabam sub-utilizados. Estes paradoxos estão nos levando a um novo modelo de computação distribuída.

Visando resolver essas e outras questões, surge a idéia de Grades de Computação (*Computational Grids*)[FK99]. O objetivo é prover maneiras de interconectar computadores e recursos computacionais diversos para estender a fronteira do que é possível realizar utilizando-se sistemas de computação. O nome deriva da rede de energia elétrica (*Power Grid*), e simboliza a esperança de que, quando bem estabelecidas, as grades permitirão acesso a recursos computacionais de maneira tão simples e onipresente quanto utilizar a rede elétrica. Da mesma forma que podemos entrar em qualquer sala do mundo e ligar um aparelho na tomada, esperamos desenvolver uma infra-estrutura que nos permitirá obter um enorme poder computacional através de uma simples ligação à Grade a partir de qualquer ponto.

1.1 Exemplos de Utilização de Grades

A idéia de Grade transcende o uso de redes que fazemos hoje em dia. Em uma rede que se encontra sob um único domínio administrativo, é comum que exista o compartilhamento de recursos: discos e impressoras, por exemplo. Mas quando a rede ultrapassa um domínio administrativo, tal compartilhamento se torna muito limitado. Esta é a finalidade das grades: permitir compartilhamento de recursos mesmo que estes estejam espalhados por diversos domínios administrativos. Os exemplos a seguir ilustram o que o uso de grades pode proporcionar.

- Cientistas de um determinado país desejam adquirir equipamentos para montar um laboratório de Física. Diversas universidades do país desejam tal laboratório, mas há dinheiro para apenas um. Utilizando grades de computação, seria possível que cientistas de universidades que não obtivessem o laboratório controlassem remotamente os instrumentos, podendo inclusive realizar experimentos à distância. São os chamados "colaboratórios"(*Collaboratories*), que permitem a colaboração entre indivíduos localizados em lugares geograficamente distantes.

- Diversas aplicações científicas (por exemplo, previsão do tempo, simulações mercadológicas, análise de prospecção de petróleo, reconhecimento de padrõoes, algoritmos de otimização) dependem de um grande poder computacional. Em muitos casos, estas aplicações podem paralelizar a sua computação de forma a se utilizar de praticamente

todos os computadores disponíveis em determinada instituição. Muitas vezes a falta de poder computacional impede trabalhos e atrasa resultados. Por outro lado, freqüentemente é inviável a compra de computadores em nível suficiente para atender às necessidades de pico. Tal situação torna-se ainda pior quando a necessidade de computação é esporádica, o que desencoraja a instituição a gastar dinheiro com algo que será usado poucas vezes. As grades podem resolver este problema, provendo acesso remoto a processadores de outras instituições que estejam ociosos naquele instante, por exemplo. No caso de grandes instituições com diversos departamentos, pode-se utilizar a grade para atender à maior necessidade computacional de um departamento através do compartilhamento do tempo ocioso das máquinas de outros departamentos.

- O crescente uso de computadores em nossa sociedade vem provocando o aumento do lixo produzido pelo descarte de equipamentos antigos. Tal problema apresenta diversas intensidades em diferentes lugares do mundo, sendo mais acentuado nos países desenvolvidos. Entretanto, com a intensificação do uso de computadores nos mais diversos lugares, a produção de sucata de informática só tende a aumentar. Entretanto, acreditamos que boa parte dos computadores descartados poderiam ser utilizados nas grades de computação. Se uma máquina já não atende às necessidades de um usuário, poderíamos juntar centenas delas para obter um poder computacional considerável.

- Grades podem permitir o acesso compartilhado a um recurso de hardware muito específico e caro. Tomemos como exemplo uma placa de compressão de vídeo MPEG-2, cujo preço pode ser equivalente ao de uma dezena de computadores tipo PC. Mesmo que este hardware seja necessário, provavelmente é inviável e até mesmo contraproducente equipar cada uma das máquinas de uma empresa ou universidade com tal placa. Assim, pode-se comprar apenas uma placa para um determinado grupo de usuários e utilizar as grades para ter acesso à placa de forma transparente, independentemente da máquina que o usuário esteja utilizando. Desta forma, uma universidade como a USP, por exemplo, poderia adquirir um servidor de geração de vídeo MPEG-2 contendo hardware altamente especializado, e o seu serviço poderia ser exportado para todos os usuários interessados em qualquer um dos campi da universidade através da Grade.

- Quando bem estabelecidas, as grades permitirão também a troca e o comércio de recursos. Duas organizações poderiam combinar a troca de poder de processamento por acesso a hardware especial, por exemplo. Também existe a possibilidade de formação de empresas que venderiam poder computacional de terceiros. Estes receberiam micropagamentos em troca dos recursos disponibilizados.

1.2 Requisitos

A adoção de grades de computação apresenta diversas vantagens. Entretanto, alguns requisitos devem ser atendidos para tornar tal tecnologia de uso corrente. Alguns deles são:

- **infra-estrutura**: para que indivíduos adotem as grades de computação, certamente

é preciso que haja uma infra-estrutura razoavelmente desenvolvida, de fácil utilização e que conte com uma base de usuários inicial;

- **padronização e interoperabilidade**: mesmo que se desenvolvam diferentes sistemas de computação em grade, é desejável que existam meios de integração dos mesmos. Caso contrário, a adoção das grades pode ser reduzida, já que a base de usuários estará pulverizada através de diversos sistemas;

- **qualidade de serviço**: os usuários exigirão das grades níveis de serviços a serem atingidos. Tolerância a falhas, disponibilidade e desempenho serão fatores cruciais;

- **custo reduzido**: o preço pago pelo acesso às grades, tais como os custos de manutenção e implementação devem ser bem inferiores ao custo de adquirir muitas máquinas, caso contrário o benefício de uso de grades não será claro.

O objetivo deste relatório técnico é introduzir o sistema InteGrade, que estamos desenvolvendo no IME-USP, e coloca-lo no contexto de outros sistemas para computação em grade já disponíveis. Assim, a Seção 2.1 descreve o sistema Globus, que permite construir aplicações em grade de maneira incremental, a Seção 2.2 apresenta o sistema Legion, que objetiva a união de recursos computacionais para formar um metacomputador, a Seção 2.3 descreve Condor, o pioneiro dos sistemas para computação em grade e a Seção 2.4 apresenta BOINC, projeto sucessor de SETI@home que busca generalizar sua aplicação a diferentes projetos. Finalmente, a Seção 3 apresenta o InteGrade, sistema que desenvolveremos, ressaltando seus princípios de construção e diferenças em relação aos demais sistemas.

2 Sistemas de Computação em Grade

Esta seção descreve quatro projetos correntes de computação em grade, a saber: Globus, Legion, Condor e BOINC.

2.1 Globus

Globus [Glo02, FK99] é um projeto que visa construir serviços de software que permitam que aplicações possam ser executadas em grades de computação. Globus é caracterizado como um conjunto de serviços para computação em grade, o que permite que aplicações para grade sejam desenvolvidas de maneira incremental, adicionando-se serviços extras com o passar do tempo. Por exemplo, pode-se primeiro utilizar ferramentas para coordenar a execução simultânea da aplicação em diversas máquinas, adicionando-se posteriormente transferências de arquivos nas grades e monitoramento.

Os serviços de Globus são divididos em serviços locais e globais. Cada serviço global é definido em termos de serviços locais. Assim, para um dado serviço, por exemplo, alocação de recursos, cada máquina ou domínio administrativo executa um serviço local gerenciador e diversos destes se unem para representar um serviço global de alocação de recursos. Cada serviço também possui uma "arquitetura ampulheta". Como as grades devem funcionar

sobre as mais diferentes arquiteturas, deve-se manter uma interface simples que seja comum a todas. Assim, os serviços locais são implementados de modo a fornecer uma interface enxuta, o centro da ampulheta, aos serviços globais.

Outra característica peculiar ao sistema Globus são as interfaces translúcidas, cujo objetivo é administrar a heterogeneidade, ao invés de escondê-la. Normalmente as interfaces são usadas para limitar as interações entre diferentes partes de um sistema, facilitando assim a sua programação e manutenção. Entretanto, como muitas aplicações a serem executadas nas grades exigem alto desempenho, pode ser desejável que alguns detalhes das camadas de baixo nível sejam visíveis e possivelmente alterados. Segundo os idealizadores, é possível adotar tal estratégia sem que isso implique em interfaces complexas.

A idéia de interfaces translúcidas é análoga às meta-interfaces de sistemas reflexivos como, por exemplo, os sistemas de middleware reflexivos [KCCB02] que utilizam o modelo de reflexão computacional para implementar uma plataforma de middleware reconfigurável.

A seguir, apresentamos uma descrição de quatro dos principais serviços do sistema Globus.

2.1.1 Gerenciamento de Recursos

Gerenciamento de recursos certamente é parte muito importante de uma grade de computação. Decidir que aplicação será executada em qual máquina pode ser fundamental para o desempenho. O serviço de gerenciamento de recursos do Globus é implementado de maneira hierárquica. Na base da hierarquia, está o gerenciador de recursos Globus (*Globus Resource Alocation Manager* ou GRAM), responsável pela administração de um conjunto local de máquinas. Este conjunto pode ser composto de uma ou mais máquinas paralelas, um *cluster* ou algumas estações de trabalho. Uma grade normalmente tem vários GRAM. Os recursos necessários a uma aplicação são expressos em termos de uma linguagem de especificação de recursos (*Resource Specification Language* ou RSL).

Um nível acima na hierarquia, existem os co-alocadores de recursos, responsáveis por gerenciar grupos de um ou mais GRAMs. Finalmente, no nível mais alto, gerenciadores de recursos específicos para cada aplicação se encarregam de distribuir as tarefas para os co-alocadores.

2.1.2 Comunicação

A comunicação no Globus é feita utilizando a biblioteca Nexus [FKT97]. Seguindo a idéia da arquitetura ampulheta, ela provê uma interface simplificada que opera sobre diversos protocolos de rede, tais como IP, ATM, ou outras formas de comunicação, como memória compartilhada e troca de mensagens. Sobre tal interface são implementados serviços de mais alto nível, como MPI, RPC e entrada e saída remota.

Uma operação de comunicação no Globus é composta de dois passos. Primeiro cria-se um canal de comunicação ligando um ponto origem e um ponto destino. Posteriormente inicia-se a comunicação aplicando-se um pedido de serviço remoto ao ponto origem. Este mecanismo transfere os dados para o ponto destino e no momento em que estão disponíveis

são integrados ao processo que requisitou a transferência. Pontos origem e destino podem ser inter-conectados de diversas maneiras, formando redes de comunicação complexas.

Uma aplicação pode utilizar diversas linhas de comunicação, aplicando em cada uma delas uma propriedade desejada. Por exemplo, pode-se escolher utilizar um protocolo não confiável para obter melhor desempenho de comunicação em uma linha, enquanto em outra pode-se decidir por um protocolo confiável.

Aqui percebe-se como as interfaces translúcidas podem ser utilizadas. Se o programador da aplicação utilizar a interface Nexus sem se preocupar com os detalhes da camada inferior, tudo funciona normalmente sobre qualquer protocolo disponível no sistema Globus. Entretanto, se ganho de desempenho deve ser obtido a todo custo, o programador pode descobrir propriedades abaixo da interface, podendo, por exemplo, trocar o protocolo de comunicação.

2.1.3 Serviço de Informação

Grades de computação são ambientes extremamente dinâmicos. Componentes podem falhar ou serem substituídos. Assim, é importante a existência de um componente que disponibilize informações sobre a situação da grade. Em Globus, tal componente é o serviço de diretórios de Meta-computação (*Metacomputing Directory Service* ou MDS). O MDS armazena informações sobre diversos aspectos da grade, como arquitetura de hardware dos nós, sistemas operacionais, memória disponível, latência e largura de banda da rede, entre outros. Mantendo o compromisso de utilizar padrões disponíveis sempre que possível, cada serviço local do MDS é simplesmente um servidor LDAP [YHK95] que armazena informações de um domínio local. O serviço MDS global é simplesmente a reunião de diversos servidores LDAP.

Cada componente de Globus é responsável por reunir informações relevantes e incluí-la no MDS. Assim, por exemplo, cada GRAM possui um módulo que recolhe informações sobre os recursos sob sua responsabilidade e atualiza o serviço MDS.

2.1.4 Serviço de Segurança

O principal desafio em um sistema de computação em grade com relação à segurança é idealizar um sistema que se adapte bem ao ambiente dinâmico e à distribuição de usuários por diferentes domínios administrativos.

A infraestrutura de segurança de Globus (*Globus Security Infrastructure* ou GSI) lida apenas com a autenticação de entidades no sistema. Um problema agravado pelas grades é que dois processos podem comunicar-se utilizando diversos mecanismos simultaneamente. A autenticação então é mais complicada do que no modelo cliente/servidor, onde quase sempre temos um único canal de comunicação entre os dois pontos. Outro problema é a heterogeneidade entre domínios administrativos, que podem possuir mecanismos diversos de autenticação. O objetivo é permitir que, uma vez autenticado, o usuário receba uma credencial que permita a ele acessar recursos sem ter que se autenticar novamente. Essa credencial seria válida por toda uma sessão.

O GSI mantém a filosofia Globus de construir serviços globais sobre serviços locais. Assim, as credenciais são mapeadas para os mecanismos locais de autenticação de cada

domínio administrativo, permitindo que diversos ambientes participem das grades sem a necessidade de alterar suas políticas locais.

2.1.5 Críticas à Arquitetura

Apesar da popularidade do sistema Globus, sua arquitetura é alvo de críticas. Os promotores do sistema Legion, descrito a seguir, argumentam [Leg02a] que a filosofia de conjunto de serviços não é expansível, isto é, acreditam que com o desenvolvimento de novos requisitos para aplicações em grade, a manutenção das aplicações existentes se tornará cada vez mais difícil. Outras deficiências que enxergamos são as seguintes:

- não possui suporte sofisticado para mobilidade de código. No entanto, existe proposta [LSM02] para implementar suporte, ainda que seja limitado à linguagem Java;
- escrito em C, não adotou o paradigma de orientação a objetos o que tende a trazer problemas de flexibilidade, extensibilidade e manutenção;
- não utiliza padrões (como por exemplo, CORBA); é baseado em protocolos próprios o que dificulta a integração com aplicações;
- não possui *sandbox*, software que limita as ações possíveis de um programa, impedindo que ele aja de maneira a causar danos a outros programas e arquivos do usuário, por exemplo;
- não se preocupa com o escalonamento local e, portanto, não tem forma de alocar recursos e garantir qualidade de serviço para seus usuários. Entretanto, existem trabalhos em desenvolvimento [FKL+99] para implementar tais funcionalidades.

2.2 Legion

Legion [Leg02b, NHG01, FK99, LG96] objetiva construir um sistema de computação em grade baseado em objetos. Sua arquitetura é bem diferente da arquitetura do Globus. Em Legion, todos os elementos da Grade são representados por objetos, sejam eles dados ou objetos reais, tais como microscópios, telescópios e outros equipamentos. Alguns dos objetivos da arquitetura são os seguintes:

- autonomia para diferentes domínios: cada domínio administrativo pode especificar como seus recursos podem ser utilizados. Legion não impõe nenhuma política aos usuários;
- núcleo extensível: a arquitetura de Legion permite que partes do sistema sejam trocadas ou aprimoradas conforme as necessidades dos usuários;
- arquitetura escalável: o sistema é totalmente distribuído de modo a permitir grande escalabilidade;

- ambiente de computação homogêneo: a grade deve esconder as diferentes plataformas sobre as quais está executando;
- espaço de nomes único e persistente;
- aproveitamento de recursos heterogêneos: em algumas situações, pode ser conveniente executar uma aplicação em uma máquina mais adequada;
- suporte a múltiplas linguagens e interoperabilidade;
- tolerância a falhas;
- executar com poucos privilégios, ou seja, não deve exigir nenhum tipo de privilégio de super-usuário.

2.2.1 Objetos e Classes

Cada objeto de Legion possui uma classe. A classe possui responsabilidades de sistema, criando, ativando e desativando objetos, assim como agendando sua execução. O usuário pode definir suas próprias classes, inclusive substituir classes padrão do sistema, caso isso seja necessário. Os objetos se comunicam por chamadas de método assíncronas, e a interface de cada classe é descrita por um tipo de IDL.

Os objetos podem estar em um de dois estados: ativo ou inerte. No estado ativo, o objeto se encontra em memória, pronto para receber chamadas de métodos. No estado inerte o objeto está seriado em algum disco da grade. Tal estratégia é utilizada pois pode ser inviável manter grande número de objetos em memória quando esta estiver escassa.

O sistema de nomes de Legion possui três níveis. No nível mais alto, os objetos são referenciados por nomes de contexto, que são cadeias de caracteres legíveis. Essas cadeias de caracteres são mapeadas para identificadores de objetos de Legion (*Legion Object Identifier* ou LOID), que, da mesma maneira que IORs em CORBA, identificam um objeto globalmente. Para efeitos de comunicação, cada LOID é mapeado para um endereço de objeto Legion (*Legion Object Address* ou LOA). Cada LOA contém uma ou mais maneiras de se comunicar com um objeto. Caso o protocolo utilizado seja IP, por exemplo, o LOA contém um ou mais pares (endereço, porta).

Legion inclui alguns objetos que implementam funções úteis a todas as outras classes de objetos. Tais objetos são chamados objetos núcleo. Estes objetos podem ser reimplementados pelo usuário caso haja necessidade, pois o sistema define apenas as interfaces necessárias. Como referência, existe uma implementação padrão de todos os objetos núcleo. Estes são:

- objetos de Contexto: mapeiam nomes de contexto (cadeias de caracteres) para LOIDs, permitindo que objetos tenham nomes arbitrários;
- agentes de Associação (*Binding Agents*): guardam pares (LOID,LOA), utilizados para localizar objetos. Os agentes de associação podem implementar *caching* e organizar-se de maneira hierárquica para aumentar a eficiência do mecanismo;

- objeto Hospedeiro: representa um processador na grade. Recebem chamadas de outras classes que requisitam a ativação de objetos na máquina que o objeto hospedeiro representa. O objetivo é mascarar as diferentes formas de criação de processos nas diferentes plataformas sobre as quais o sistema funciona;

- objeto Cofre: representa um parte de um disco, mas apenas para propósitos de representação de objetos inertes, ou seja, seriados;

- objeto de Implementação: representa um programa a ser executado. Quando um objeto hospedeiro recebe uma requisição para ativar ou criar um objeto, o objeto de implementação é transferido para o objeto hospedeiro, o que permite que este crie um processo que execute o programa desejado.

2.2.2 Segurança

Em relação à segurança, os objetivos primários de Legion são deixar que o usuário instale o sistema sem que isso implique em riscos à segurança e permitir a configuração dos sistemas de segurança de modo a permitir que o usuário escolha as políticas mais adequadas às suas necessidades. Como Legion pode ser executado com poucos privilégios, o usuário pode limitar as ações que as aplicações podem executar. Além disso, a arquitetura expansível e modular permite que pedaços do sistema possam ser trocados caso seja necessário aumentar a segurança. Tudo depende do balanço entre conveniência de uso e segurança, decisão deixada ao cargo dos usuários.

O modelo de objetos de Legion provê funcionalidades que permitem implementar políticas de segurança. Como todas as entidades do sistema são representadas por objetos, a política de segurança pode ser associada à permissão de chamada dos métodos. Assim, um objeto pode permitir que outro chame seus métodos, apenas parte deles ou nenhum.

Cada objeto Legion possui por definição um método chamado "MayI", que indica a um usuário os métodos do objeto ao quais ele tem acesso. Quando um usuário ou objeto adquire os direitos de chamada de método a outro objeto, recebe um certificado listando os seus direitos. Através do uso de criptografia, esse certificado é praticamente inviolável, não podendo ser alterado ou forjado. Ao fazer uma chamada ao objeto emissor do certificado, o método "MayI" intercepta a chamada e, dado o certificado, verifica se tal usuário possui os direitos de invocar tal método. Caso a chamada seja permitida, tudo procede conforme o esperado. Caso contrário, a chamada não se realiza.

Cada objeto Legion possui um par de chaves criptográficas. A chave pública é embutida no seu identificador de objeto (LOID), e permite que outros objetos a utilizem para criptografar as comunicações entre ambos. A chave privada permite que um objeto assine suas mensagens.

2.2.3 Adaptabilidade e Suporte a Alto Desempenho

O sistema Legion, além de prover um ambiente para processamento paralelo, também permite que tarefas sejam designadas a nós mais adequados a executá-la. Pode-se utilizar um

Benchmark ou a própria aplicação para determinar quais as condições mais adequadas à execução da aplicação.

Legion provê suporte a aplicações escritas em MPI ou PVM, bastando que estas sejam recompiladas e religadas para que funcionem sobre a grade. Algumas linguagens como MPL (Mentat Programming Language, extensão de C++), BFS (Basic Fortran Support) e Java podem ser utilizadas no desenvolvimento de aplicações Legion.

Aplicações legadas podem ser encapsuladas dentro de um objeto. Assim, aplicações PVM, por exemplo, são encapsuladas em objetos Legion e com isso podem ser acessadas por outros objetos.

Além disso, são disponibilizadas bibliotecas que permitem que se estenda o sistema em termos de linguagens disponíveis.

2.2.4 Escalonamento e Gerenciamento de Recursos

O escalonamento no sistema Legion é baseado em dois pré-requisitos: os proprietários dos recursos devem poder determinar como e quando os recursos podem ser usados. Impor políticas aos provedores implicaria na resistência à adoção do sistema. Também os usuários devem poder especificar os recursos de que necessitam, de modo a não sacrificar o desempenho de suas aplicações. Em alguns casos, pode ser que o usuário deseje escolher políticas específicas de escalonamento ou até projetar escalonadores específicos para determinada aplicação.

2.3 Condor

O sistema Condor [Con02, LLM88, FK99, CT01] certamente é o mais antigo dos aqui apresentados. Sua origem data de 1985, e foi derivado de outro projeto desenvolvido na Universidade de Wisconsin chamado *Remote Unix*. Dos três projetos apresentados, Condor é o que dá mais ênfase à utilização de recursos ociosos em redes de computadores, que é o foco principal do InteGrade.

O usuário alvo de Condor é aquele que possui necessidade de grande poder computacional ao longo de grandes unidades de tempo, tais como dias, meses ou anos. Normalmente o conjunto de dados a ser processado é muito maior que o tempo disponível para fazê-lo; assim o tempo disponível acaba determinando o quanto será processado. A tal característica de processamento dá-se o nome de *High Troughput Computing* ou HTC [LBRT97]. Note que HTC contrasta com computação de alto desempenho (HPC), que normalmente se preocupa com desempenho ao longo de curtos períodos de tempo, utilizando medidas de desempenho como FLOPs.

Com o passar dos anos, os recursos computacionais que antes eram centralizados passaram a ser distribuídos pelas instituições. Atualmente, cada membro possui uma máquina, na qual desempenha suas funções diárias. Entretanto, durante grande parte do tempo essas máquinas ficam ociosas. Recursos fragmentados representam comodidade de uso, mas também incorrem em desperdício de poder computacional. Como os usuários de HTC precisam de quase todo o poder computacional disponível, fica evidente que qualquer recurso que possa ser utilizado é bem vindo aos sistemas de HTC, e esse é o objetivo de Condor.

2.3.1 Gerenciamento de Recursos

As tarefas de gerenciamento de recursos de um sistema Condor devem atender a quatro grupos de interesse:

- **proprietários de recursos**: devem ter suas políticas de compartilhamento sempre respeitadas. Não devem sentir degradação no desempenho de suas aplicações, caso contrário relutarão em fornecer seus recursos para a grade;
- **gerenciadores de recursos**: desejam estar confiantes em relação à segurança e robustez do sistema. Não devem ter de interferir muito para o funcionamento;
- **programadores de aplicações**: desejam uma interface de programação fácil de ser entendida e embutida em suas aplicações;
- **usuários**: precisam ter suas necessidades atendidas. Não adianta poder acessar recursos de terceiros se o seu trabalho não é executado de maneira aceitável.

Para atender tais objetivos, Condor utiliza uma arquitetura em camadas, cada qual desempenhando uma função para o gerenciamento de recursos. São elas:

- **camada local**: é a camada mais baixa de gerenciamento de recursos, podendo ser fornecida pelo sistema operacional ou até por outra grade;
- **camada do proprietário**: interage com as restrições de uso do proprietário dos recursos, oferecendo para o sistema apenas aqueles realmente disponíveis. Dessa forma, se o proprietário especifica que entre as 7 e 8 horas da manhã nada pode executar em sua máquina, tal camada não fornece esses recursos para as camadas superiores;
- **camada de sistema**: responsável pelo emparelhamento entre a oferta e a procura de recursos;
- **camada do usuário**: representa as necessidades dos usuários, implementando suas requisições como uma fila de pedidos e administrando tal fila de maneira persistente e tolerante a falhas;
- **camada de gerenciamento de recursos da aplicação**: quando uma aplicação consegue obter recursos, estes são passados para a camada de gerenciamento de recursos da aplicação, que é responsável pela comunicação com tais recursos. Também provê a possibilidade de se requisitar recursos adicionais no decorrer da execução, o que pode ser útil para a construção de aplicações adaptativas;
- **camada de aplicação**: representa as tarefas da aplicação do usuário.

2.3.2 Alocação de Recursos

O sistema Condor usa um mecanismo de emparelhamento entre oferta de recursos e procura dos mesmos. Tal processo é composto por três fases:

1. **descrição da oferta e procura**: tanto o proprietário do recurso quanto o usuário devem informar ao sistema quais as suas restrições para a execução de aplicações. Cada um deles elabora um anúncio, em analogia aos anúncios classificados, que determina os requisitos e restrições. Assim, por exemplo, o dono de uma estação de trabalho pode colocar no anúncio que só aceita iniciar a execução de processos remotos quando a máquina estiver inativa por 15 minutos ou à noite. Já o usuário dos recursos pode especificar as características necessárias à execução de sua aplicação, como plataforma, quantidade de memória, etc. Pode também informar as características desejáveis mas não imprescindíveis;

2. **emparelhamento entre a oferta e a procura**: com os diversos anúncios no sistema, entra em cena o serviço de emparelhamento. Sua tarefa é analisar os diversos anúncios presentes no sistema e emparelhá-los de modo a atender às necessidades dos usuários e às restrições dos proprietários de recursos. Pode-se possuir vários algoritmos que podem emparelhar baseado em critérios diferentes, como justiça ou prioridade, por exemplo;

3. **atribuição dos recursos**: após o emparelhamento, as partes devem decidir se aceitam ou não iniciar a computação. Como o ambiente de uma grade é dinâmico, pode ser que o anúncio de um recurso já não seja totalmente válido; por exemplo, pode ocorrer que a quantidade de memória disponível tenha diminuído no intervalo entre a publicação do anúncio e o emparelhamento. Assim, nessa fase as partes trocam informações para decidir se prosseguem ou não a computação. Caso as condições sejam inaceitáveis, o ciclo se reinicia.

2.3.3 *Checkpointing*

Devido ao ambiente dinâmico de uma grade, a disponibilidade de recursos é algo muito variável. Uma máquina pode ficar ociosa por alguns minutos ou horas. Assim, é muito difícil prever se a aplicação conseguirá terminar antes do dono do recurso voltar ao uso de sua máquina. Para evitar a perda de computação já realizada, Condor inclui mecanismos de *checkpointing*. Quando uma máquina torna-se indisponível, todo o estado da aplicação que está sendo executado é gravado em disco ou transferido pela rede para outra máquina. Isso inclui também as comunicações de rede pendentes. Nesse caso, o *buffer* de rede é esvaziado e a aplicação é migrada. Ao chegar em uma nova máquina, seu estado é restaurado, assim como os canais de comunicação. Uma aplicação também pode requisitar um *checkpoint* explicitamente, ou impedir um *checkpoint* se estiver em uma região crítica.

Checkpointing de aplicações paralelas ainda não é suportado por Condor. Entretanto, existem pesquisas que objetivam adicionar tal funcionalidade, o que seria extremamente útil para o sistema, permitindo a execução de aplicações paralelas inclusive em recursos não dedicados.

2.3.4 Utilização Incremental de Condor

O sistema Condor possui diferentes maneiras de utilização, que podem ser organizadas de maneira hierárquica. São elas:

- utilizar Condor em uma só máquina (*Personal Condor*): tal idéia pode parecer de utilidade limitada, mas mesmo assim é bem útil quando uma aplicação deve ser executada várias vezes, possivelmente com diversos parâmetros diferentes. Nesse caso, Condor pode manter um registro das execuções, avisar sobre o estado corrente da execução e implementar uma política de execução da aplicação. Também pode utilizar o mecanismo de *Checkpointing* para implementar tolerância a falhas de maneira transparente;

- utilizar Condor em um domínio administrativo (*Condor Pool*): nesse caso tudo prossegue como já descrito. Caso as máquinas devam atender a diferentes usuários com diferentes prioridades, deve-se preencher os anúncios de modo a obter o resultado desejado;

- utilizar Condor em vários domínios administrativos (*"Flocking"*): caso haja acordo entre dois ou mais domínios administrativos para o compartilhamento dos recursos, pode-se configurar Condor de maneira a permitir que processos de um domínio execute em outro. A política de execução flexível permite que se aplique restrições à execução de processos não locais ou priorizar os processos locais, por exemplo;

- utilizar Condor para acessar grades Globus: utilizando Condor-G, é possível acessar grades Globus e executar as tarefas em outra grade. Utilizando GlideIn, é possível adquirir recursos de uma grade Globus de maneira temporária para executar determinada aplicação. Após a execução, os recursos retornam à grade Globus.

2.4 BOINC

BOINC [BOI02] é um projeto da Universidade da California em Berkeley, mesmos desenvolvedores de SETI@home [SET02], projeto de computação em grade de maior repercussão de todos os tempos, principalmente fora da comunidade acadêmica. Apesar de bem sucedido, SETI@home possui diversas limitações que são sanadas por BOINC.

A maior deficiência de SETI@home é a impossibilidade de utilizá-lo para resolver diversos problemas. O único problema que utiliza o sistema é o SETI (*Search for Extraterrestrial Intelligence*), e este não pode ser trocado por outro. Tal inconveniente é resolvido por BOINC, uma vez que ele provê um arcabouço que pode ser utilizado por diversas aplicações.

O proprietário de uma máquina que decida oferecer seus recursos ao sistema BOINC possui maior flexibilidade no controle do uso de seus recursos, podendo especificar a quantidade de cada recurso que deseja compartilhar, tais como processador, memória, disco e largura de banda.

Assim como SETI@home, BOINC possui como característica marcante a intenção de incentivar que usuários domésticos proprietários de equipamentos comuns incorporem seus recursos à grade. Para incentivar tal adesão, oferece um protetor de tela visualmente atraente

que é exibido quando o sistema está em funcionamento. Tal característica é um grande mérito de SETI@home e BOINC, uma vez que a quantidade de computadores espalhados pelas residências ao redor do Globo é enorme, e grande parte se encontra ociosa em qualquer instante de tempo.

Os desenvolvedores de aplicações que desejam resolver problemas intensivos em computação escrevem código de aplicação que constituirá um projeto. BOINC oferece uma API em C++ para desenvolvimento da aplicação que oferece acesso a recursos do sistema, como *checkpointing*.

Aplicações podem possuir diversos executáveis, operando em esquema mestre-escravo. Entretanto, para a aplicação fazer melhor uso do sistema, é necessário que cada um dos nós da aplicação possua pouca ou nenhuma comunicação com os demais.

BOINC possui muitas afinidades com InteGrade, tais como utilização de equipamentos comuns e controle total do proprietário sobre seus recursos. Entretanto, características como padrões de acesso e suporte mais genérico a aplicações paralelas e a objetos distribuídos, presentes no InteGrade, não são contemplados no momento por BOINC.

3 InteGrade

O projeto InteGrade, em fase inicial de desenvolvimento no IME-USP, visa construir uma infra-estrutura genérica de middleware que permita a utilização do tempo ocioso das máquinas já disponíveis em instituições públicas e privadas para a resolução de diversos problemas altamente paralelizáveis.

A importância do InteGrade é evidenciada quando analisamos as necessidades de um país carente de recursos, como o Brasil. Ao passo que pesquisadores necessitam de recursos computacionais robustos e caros, na própria instituição já existem dezenas ou centenas de máquinas sub-utilizadas. O tempo ocioso de tais máquinas pode ser usado pelos pesquisadores, resultando em economia de recursos e eliminação de desperdício.

InteGrade será construído utilizando o que existe de mais avançado em sistemas de objetos distribuídos, padrões da indústria e protocolos de computação distribuída de alto desempenho.

Os principais requisitos que serão considerados no desenvolvimento de InteGrade são os seguintes.

- **O sistema deve se auto-conhecer**: implica na necessidade da manutenção de uma base de dados atualizada dinamicamente que contenha informações sobre as máquinas que fazem parte do sistema, plataformas de hardware e software de cada máquina, tipo de rede local que interliga as máquinas de um agrupamento, tipo de rede de média e longa distância que interliga os diversos agrupamentos, além do estado dinâmico da grade, isto é, quantidades disponíveis de recursos como disco, processador, memória e largura de banda.

- **Sobrecarga quase imperceptível pelos clientes**: o middleware deverá ser capaz de aproveitar os recursos ociosos disponíveis nas máquinas dos clientes com o menor impacto possível no desempenho percebido pelos usuários das máquinas clientes. Ou

seja, sempre que o usuário utilizar a sua máquina, ele deverá ter prioridade máxima no uso dos recursos e a sobrecarga imposta pelo middleware deverá ser imperceptível. Caso contrário, será mais difícil obter o consentimento dos usuários para integrar a Grade.

- **Garantias de segurança:** já que será possível carregar dinamicamente código executável nas máquinas do cliente, é importante garantir que tal código não prejudicará o correto funcionamento de outras aplicações sendo executadas na máquina do cliente e que este novo código não irá modificar ou mesmo ter acesso a informações pessoais, possivelmente confidenciais, armazenadas nas máquinas dos clientes.

3.1 Diferenciais do InteGrade em relação a outros projetos de Grades

- **Reutilização da base instalada**: é um princípio chave do projeto InteGrade. Tal característica também pode ser observada nos outros projetos já descritos, porém de maneira mais discreta em Globus e Legion, e mais acentuada em Condor. O diferencial é que InteGrade considerará essa característica no desenvolvimento de sua arquitetura. Além disso, há uma tendência à utilização de recursos dedicados, sobretudo para computação paralela. InteGrade não exigirá recursos dedicados, mas poderá utilizá-los caso esta seja a vontade dos proprietários de recursos.

- **Utilização de tecnologias modernas de objetos distribuídos**: característica única do InteGrade, que utilizará CORBA na sua construção. Condor e Globus são escritos em C, ao passo que Legion utiliza MPL, variante de C++, mas não CORBA na sua construção.

  Com isso, obtemos duas vantagens: (1) poderemos re-utilizar os inúmeros serviços já disponíveis na arquitetura CORBA e (2) será mais fácil a integração de outros serviços e aplicações à grade dado que esta utilizará uma tecnologia padrão de interconexão orientada a objetos.

- **Baixa sobrecarga**: como já descrito, o middleware do InteGrade vai priorizar o proprietário do recurso. Assim, é necessário que o sistema não consuma muitos recursos de maneira a não se tornar um problema para o usuário. Em particular, pretendemos utilizar ORBs CORBA que consumam poucos recursos, como UIC-CORBA, descrito em [KRC01].

3.2 Linhas de Pesquisa

A equipe de pesquisadores deste projeto foi composta de forma a incluir especialistas em todas as áreas essenciais descritas acima. Em particular, a equipe possui uma larga experiência nas áreas de algoritmos paralelos e paralelização de código, middleware CORBA e Java, agentes móveis, gerenciamento de recursos em sistemas distribuídos e qualidade de serviço em comunicação na Internet. A seguir, apresentamos as seis linhas de pesquisa que comporão o projeto.

3.2.1 Requisitos e Arquitetura Geral do Sistema

O objetivo dessa linha de pesquisa é fazer um levantamento dos requisitos do sistema, definir uma arquitetura de referência inicial e especifica-la através de diagramas UML. Alguns pontos iniciais a serem considerados são:

- serviços do sistema: definição dos serviços necessários às aplicações que executarão na grade. Sabemos da necessidade de um serviço de informação e de um serviço de escalonamento. Outros serviços, como contabilização de recursos utilizados, poderão ser definidos em uma fase posterior de pesquisa;

- formato do software a ser utilizado: consideramos a necessidade inicial de dois programas a serem instalados nas máquinas integrantes da grade. O fornecedor de recursos, isto é, indivíduo que concorda em ceder parte do tempo ocioso de sua máquina, executará um *daemon* que permitirá que aplicações da grade se beneficiem de seus recursos. Este usuário terá ampla possibilidade de determinar o que, quanto e como deseja compartilhar. Do outro lado está o usuário da grade, que escreve aplicações e as submete ao sistema. Este precisará de um programa que o permita submeter aplicações à Grade, assim como mecanismos de descrição dos recursos necessários, tais como: plataforma de hardware e software, processador e memória, por exemplo.

3.2.2 Paralelização de Problemas Computacionalmente Difíceis

Esta linha de pesquisa objetiva criar um arcabouço e interfaces genéricas que serão utilizados para implementar a maior gama possível de algoritmos paralelos. Alguns pontos a serem considerados são listados abaixo.

- Topologia: por um lado o uso de uma arquitetura do tipo coordenador/escravos pode se adaptar para aplicações simples e com pouca comunicação entre os nós. Entretanto, este tipo de implementação não é escalável, sendo adequado apenas para aplicações altamente paralelizáveis. Como alternativas temos estruturas hierárquicas que podem ser escaláveis, ou o modelo genérico onde pode-se permitir a comunicação entre os nós da aplicação.

- Quantidade de comunicação: Internet tradicional permite resolver problemas altamente paralelizáveis com pouca comunicação. Já a Internet2 permitirá a aplicação deste modelo para resolver também problemas que exigem que uma grande quantidade de dados sejam trocados entre os nós do sistema distribuído.

- Tolerância a falhas: a distribuição/paralelização do algoritmo deve ser feito de uma forma tolerante a falhas. Se uma das máquinas cai ou se torna inacessível, não podemos perder toda a computação que realizamos nas outras máquinas. Entretanto, o *checkpointing* paralelo é muito complexo e pode criar sobrecargas proibitivas. Pode-se resolver problemas deste tipo usando modelos para computação paralela que impõem sincronizações periódicas, como o BSP [Val90].

- Uma restrição importante do InteGrade é a prioridade absoluta ao proprietário do recurso. Caso a máquina onde uma aplicação seqüencial esteja sendo executada seja requisitada, a aplicação pode ser migrada sem grandes prejuízos, sendo que eles podem ser determinados pela política de *checkpoint*. No caso de aplicações paralelas com comunicação ponto a ponto, além dos problemas de *checkpoint* também existem problemas inerentes a possíveis migrações, como por exemplo: como tratar as comunicações em curso? Como tratar as comunicações futuras? Outra alternativa é tentar evitar casos onde as migrações sejam necessárias usando padrões de acesso (3.2.4) e técnicas de escalonamento.

- Nível de abstração: existe um compromisso claro entre o nível de abstração de uma aplicação paralela e o seu desempenho em uma máquina ou rede específica. Quanto mais detalhes são considerados melhor é o desempenho. Entretanto caso haja mudanças na máquina, todo o processo de adaptação deve ser refeito. Esta é uma grande preocupação do InteGrade, onde dificilmente se conhecem as máquinas e sua rede de comunicação disponível a uma aplicação, a priori. Na busca do compromisso podem ser usados modelos como o BSP [Val90] e o de *Malleable Tasks* [GM99].

3.2.3 Segurança

Questões como controle de acesso, criação de papéis e usuários da Grade, autenticação dos usuários e do código móvel que transita pela Grade, criptografia de dados confidenciais que transitam pela Grade serão aqui tratadas.

3.2.4 Identificação de Padrões de Acesso de Usuários e de Disponibilidade de Recursos

Para a identificação de padrões de acesso devemos desenvolver um processo de *aprendizado constante* [Mit97, LM95]. Tal processo tem início com o monitoramento constante dos seguintes recursos básicos da máquina:

- quantidade de memória disponível
- quantidade da área de *swap* disponível
- quantidade de disco disponível
- percentagem da CPU ociosa

O monitoramento consiste em coletar séries temporais destas grandezas coletadas a intervalos regulares, digamos, de 5 em 5 minutos. Posteriormente, estas séries temporais serão divididas em intervalos constantes, digamos de 24h, que se tornarão os objetos de classificação [Han97] . Chamaremos estes objetos de *períodos*.

A partir deste monitoramento pretendemos, numa primeira fase, gerar *categorias iniciais de comportamento*. Tais categorias serão geradas a partir de algoritmos de *clustering* [JW83] que agrupam *períodos* a partir das semelhanças entre os mesmos. No nosso caso, cada *período*

será um vetor de um número fixo N de pontos e uma possível medida de similaridade seria a distância euclidiana N-dimensional. Como o algoritmo de *clustering* é caro, poderemos investigar algumas das aproximações utilizadas na literatura.

Gostaríamos que o nosso método de aprendizado fosse capaz de aprender categorias como: dia típico, domingo, feriado, trabalho intenso, trabalho leve, etc.

Terminada a fase de classificação inicial, o monitoramento continua, e cada período passa a ser monitorado quanto a pertinência a alguma das categorias aprendidas. O processo de *aprendizado constante* consiste em analisar dois fenômenos:

- *Deriva de categorias.* Trata-se de identificar uma variação no *elemento prototípico* de uma dada categoria. Novos períodos são classificados dentro de uma categoria existente, mas nota-se uma variação, aqui chamada de deriva, nas características desta categoria.

- *Criação de novas categorias.* Trata-se de encontrar objetos (novos períodos) semelhantes que não se enquadram bem em nenhuma categoria, ou seja, cuja distância aos elementos prototípicos destas categorias é "muito grande".

A tipificação destes fenômenos é uma das contribuições desta linha de pesquisa. Outro fenômeno evolutivo que pode ser estudado é a morte de categorias, que pode ser considerado como uma redução no número de categorias a serem analisadas para a classificação, para evitar a deterioração do método com a criação de inúmeras novas categorias.

3.2.5 Mobilidade de Código e Computação Ubíqua

Essa linha de pesquisa objetiva estudar questões decorrentes da necessidade de código móvel no sistema InteGrade. Pontos a serem considerados são listados abaixo.

- *Sandbox* Java, que objetiva limitar a capacidade do código móvel que executa na máquina de um usuário que cede recursos, com o objetivo de proteger a integridade de seus arquivos e sistema.

- *Sandbox* Janus [GWTB96] para código nativo, que objetiva o mesmo do item anterior, porém podendo ser utilizada com qualquer programa compilado para código nativo.

- Uso da biblioteca DSRT [NhCN98] (Dynamic Soft Real Time), para limitar a quantidade de processador e memória utilizada pelo *daemon* que será executado em todas as máquinas que fornecem recursos à Grade.

Uma motivação adicional seria usar o nosso middleware para computação ubíqua. Usuários móveis poderiam utilizar seus PDAs e computadores móveis para interagir com os dispositivos computacionais fixos no espaço físico local de forma a utilizar os recursos lá disponíveis para executar as partes pesadas do seu código. O PDA poderia exportar código móvel para um PC local e usar a CPU mais potente do PC e o seu espaço em disco para executar as componentes mais pesadas.

4 Conclusões

Os sistemas de computação em grade representam um avanço nos métodos de integração de recursos computacionais. Se tais sistemas forem difundidos e adotados em grande escala, poderão representar uma extensão da fronteira do que é possível hoje em computação. Paralelamente a isso, diversas ciências se desenvolverão, uma vez que os grandes desafios científicos da atualidade demandam o uso intensivo de computadores.

Os cinco sistemas aqui apresentados possuem várias semelhanças mas arquiteturas e focos diferentes. Globus, com seu princípio de pacote de serviços, possui adoção mais rápida devido à possibilidade de inclusão incremental de serviços de grade. Entretanto, essa metodologia poderá deixar as aplicações excessivamente complexas. Legion possui arquitetura baseada em objetos e um arcabouço comum para a implementação de seus serviços. Já Condor é uma opção que pode ser adotada de maneira incremental. BOINC objetiva utilizar o êxito de SETI@home para a resolução de outros problemas. InteGrade pretende utilizar metodologias e tecnologias modernas de projeto de software em sua construção visando obter um sistema compacto, flexível e adaptável que dê acesso a uma quantidade enorme de recursos hoje ociosos sem afetar a qualidade do serviço obtida pelos seus proprietários.

Referências

[BOI02] BOINC. Sítio do projeto boinc, 2002. `http://boinc.berkeley.edu`.

[Con02] Condor. Sítio do projeto Condor, 2002. `http://www.cs.wisc.edu/condor/`.

[CT01] Peter Couvares and Todd Tannenbaum. Condor Tutorial at First Euroglobus Workshop. Disponível em: `http://www.cs.wisc.edu/condor/slides/`, 2001.

[FK99] Ian Foster and Karl Kesselman. *The Grid: Blueprint for a new Computing Structure*, chapter 2,3,5,11. Morgan Kaufmann Publishers, 1999.

[FKL+99] Ian Foster, Carl Kesselman, Craig Lee, Bob Lindell, Klara Nahrstedt, and Alain Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *International Workshop on Quality of Service*, 1999.

[FKT97] Ian Foster, Carl Kesselman, and Steve Tuecke. Managing Multiple Communication Methods in High-Performance Networked Computing Systems. *J. Parallel and Distributed Computing*, pages 40:35–48, 1997.

[Glo02] Globus. Sítio do projeto Globus, 2002. `http://www.globus.org`.

[GM99] Denis Trystram Gregory Mounie, Christophe Rapine. Efficient Approximation Algorithms for Scheduling Malleable Tasks. In *SPAA*, 1999.

[GWTB96] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer. A Secure Environment for Untrusted Helper Applications: Confining the Wiley Hacker. In *Proceedings of the USENIX Security Symposium*, July 1996.

[Han97] David J. Hand. *Construction and Assessment of Classification Rules*. John Wiley and Sons, 1997.

[JW83] Richard A. Johnson and Dean Wichern. *Applied Multivariate Statistical Analysis*. Prentice-Hall, 1983.

[KCCB02] Fabio Kon, Fábio Costa, Roy Campbell, and Gordon Blair. The Case for Reflective Middleware. *Communications of the ACM*, 45(6):33–38, June 2002.

[KRC01] Fabio Kon, Manuel Román, and Roy H. Campbell. Reflective middleware: From your desk to your hand. *Distributed Systems Online*, Vol. 2(No. 5), 2001.

[LBRT97] Miron Livny, Jim Basney, Rajesh Raman, and Todd Tannenbaum. Mechanisms for High Throughput Computing. *SPEEDUP Journal*, Vol. 11(No. 1), June 1997.

[Leg02a] Legion. Perguntas Frequentes sobre o Legion. `http://legion.virginia.edu/FAQ.html#globus`, 2002.

[Leg02b] Legion. Sítio do projeto Legion. `http://www.cs.virginia.edu/~legion/`, 2002.

[LG96] Michael J. Lewis and Andrew Grimshaw. The Core Legion Object Model. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society Press, August 1996.

[LLM88] Michael Litzkow, Miron Livny, and Matt Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages pages 104–111, June 1988.

[LM95] Pat Langley and Michael B. Morgan, editors. *Elements of Machine Learning*. Morgan Kaufmann, 1995.

[LSM02] Gregor von Laszewski, Kazuyuki Shudo, and Yoichi Muraoka. Grid-based Asynchronous Migration of Execution Context in Java Virtual Machines. Submitting, 2002.

[Mit97] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[NhCN98] Klara Nahrstedt, Hao hua Chu, and Srinivas Narayan. QoS-aware Resource Management for Distributed Multimedia Applications. *Journal of High- Speed Networking*, 7:227–255, 1998. Special Issue on Multimedia Networking.

[NHG01] Anand Natrajan, Marty Humphrey, and Andrew Grimshaw. Grids: Harnessing Geographically-Separated Resources in a Multi-Organisational Context. Presented at High Performance Computing Systems, June 2001.

[SET02] SETI@home. Sítio do projeto SETI@home, 2002. `http://setiathome.ssl.berkeley.edu/`.

[Val90] Leslie G. Valiant. A bridging model for parallel computation. *Communnications of the ACM*, 33:103–111, 1990.

[YHK95] W. Yeong, T. Howes, and S. Kille. Lightweight Directory Access Protocol. RFC #1777, March 1995.