Grids Computacionais: Arquiteturas, Tecnologias e Aplicações

Walfredo Cirne

Departamento de Sistemas e Computação

Universidade Federal de Campina Grande

http://walfredo.dsc.ufcg.edu.br

Resumo

A impressionante melhoria de desempenho que redes de computadores vêm experimentando levou a idéia de se utilizar computadores independentes conectados em rede como plataforma para execução de aplicações paralelas, originando a área de Computação em Grid. Os principais atrativos desta idéia são a possibilidade de alocar uma enorme quantidade de recursos a uma aplicação paralela (e.g. centenas de milhares de computadores conectados via Internet) e fazê-lo a um custo muito menor do que alternativas tradicionais (baseadas em supercomputadores paralelos).

As vantagens potenciais da Computação em Grid têm atraído muita atenção para a área. Nos últimos anos, grandes projetos (como Globus) foram iniciados para materializar a visão de Grids Computacionais e aplicações foram desenvolvidas (das quais, a mais conhecida é SETI@home). Mais recentemente, empresas (como Entropia e distributed.net) foram fundadas para vender serviços de Computação em Grid.

É devido ao grande potencial da área e ao interesse que ela vem recebendo em todo o mundo, que julgamos interessante a preparação deste texto. Nossa maior meta é desmistificar o tema, mostrando que Grids Computacionais são uma nova plataforma para execução de aplicações paralelas. Obviamente, Grids apresentam características diferentes das plataformas existentes. Devido a sua heterogeneidade, compartilhamento e complexidade, Grids apresentam, em geral, maiores dificuldades para execução de aplicações paralelas que plataformas tradicionais. O quão apropriado é o uso de um Grid Computacional depende em grande medida da aplicação a ser executada.

Após a discussão das características de Grids e das aplicações que os utilizam, apresentaremos os principais problemas encontrados para utilização efetiva de Grids Computacionais, bem como soluções propostas para estes problemas. Alguns sistemas para construção de Grids Computacionais também serão apresentados. O texto termina com um pequeno exercício de futurologia ao apontar as tendências para o desenvolvimento da área.

Sumário

1 Conceituação

Grids Computacionais são uma área recente e em franca expansão. O que começou em universidades e institutos de pesquisa ganhou o mundo empresarial e hoje faz parte da estratégia de corporações como IBM, HP/Compaq, Sun e Fujitsu. Em suma, Grids Computacionais são hoje um assunto em moda.

Mas, o que afinal vem a ser um Grid Computacional? A visão original estabelece uma metáfora entre A Rede Elétrica (The Electric Grid) e O Grid Computacional (The Computational Grid). A Rede Elétrica disponibiliza energia elétrica sob demanda e esconde do usuário detalhes como a origem da energia e a complexidade da malha de transmissão e distribuição. Ou seja, se temos um equipamento elétrico, simplesmente o conectamos na tomada para que ele receba energia. O Grid Computacional, portanto, seria uma rede na qual o individuo se conecta para obter poder computacional (ciclos, armazenamento, software, periféricos, etc). A Figura 1 ilustra esta idéia.

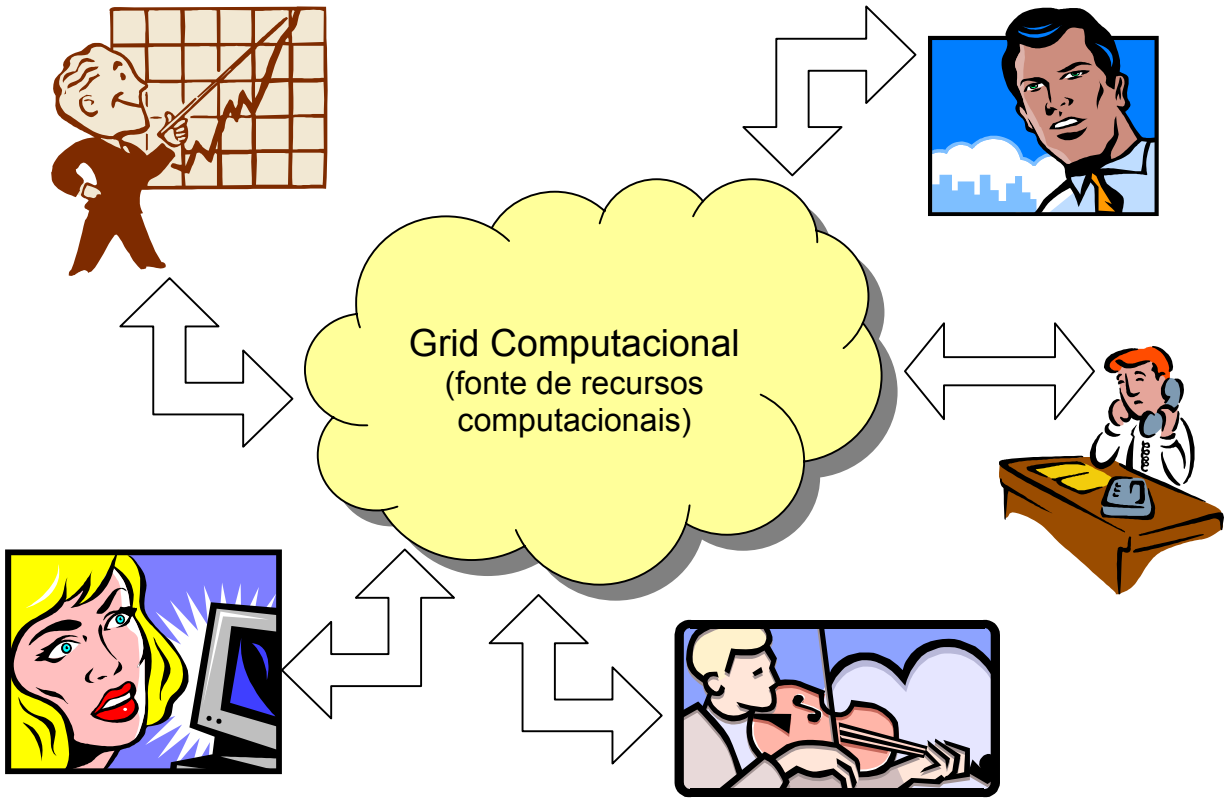

Figura 1 – O Grid Computacional como fonte transparente de poder computacional sob demanda

Um sistema que forneça poder computacional sob demanda e transparentemente é certamente desejável. Note que, para muita gente, a Internet é este sistema. De fato, para aqueles cujas necessidades de processamento são satisfeitas por um computador desktop, a Internet atende os requisitos básicos de um Grid Computacional. Por exemplo, quando usamos home banking, nosso computador desktop, uma série de roteadores e os computadores do nosso banco se agregam sob demanda para nos fornecer um serviço de forma transparente.

Quero com isso sugerir que Grid Computing é o mesmo que Internet? Bem... não!!! A visão que coloca o Grid como o equivalente computacional da Rede Elétrica conta apenas parte da história. Grids Computacionais nasceram da comunidade de Processamento de Alto Desempenho (PAD). A impressionante melhoria de desempenho que redes de computadores vêm experimentando naturalmente levou a idéia de se utilizar computadores independentes conectados em rede como plataforma para execução de aplicações paralelas [17]. Os atrativos desta idéia são a possibilidade de alocar uma enorme quantidade de recursos a uma aplicação paralela (e.g. centenas de milhares de computadores conectados via Internet) e fazê-lo a um custo muito menor do que alternativas tradicionais (baseadas em supercomputadores paralelos). Portanto, embora a metáfora do Grid oferecendo poder computacional como a Rede Elétrica oferece energia seja interessante e motivadora, ela não passa disso, de uma metáfora. Para oferecermos um conceito mais útil tecnicamente, precisamos pensar no Grid como *uma plataforma para execução de aplicações paralelas*.

Mas, certamente, há plataformas para execução de aplicações paralelas que não são Grids. O que diferencia um Grid de um cluster, por exemplo? De maneira geral, podemos dizer que Grids são mais distribuídos, diversos e complexos que outras plataformas. Aspectos que evidenciam esta distribuição, diversidade e complexidade são:

- Heterogeneidade (nos componentes do Grid),
- Alta dispersão geográfica (Grids podem ter escala mundial),
- Compartilhamento (no sentido de que o Grid não pode ser dedicado a uma aplicação),
- Múltiplos domínios administrativos (Grids podem congregar recursos de várias instituições), e
- Controle distribuído (tipicamente não há uma única entidade que tenha poder sobre todo o Grid).

Note que esta discussão propõe um *conceito* e não uma *definição* para Grid Computacional. Uma plataforma para execução de aplicações paralelas que apresenta as características acima listadas certamente é um Grid. Mas a ausência de alguma das características não deve automaticamente desqualificar uma determinada plataforma como Grid.

Por outro lado, o leitor deve estar atento ao uso do termo Grid Computacional tão e somente como ferramenta de marketing [20]. Devido a sua popularidade e a seu impacto positivo, a tendência é que o termo Grid Computing seja utilizado de forma muito “liberal”, como no passado o foram Orientação a Objetos, Sistemas Abertos, Downsizing, Reengenharia, Internet/Intranet/Extranet, entre outros.

2 Grids como Plataformas de Execução

Uma *aplicação paralela* é composta por várias *tarefas*. As tarefas que compõem uma aplicação paralela executam em vários processadores, caracterizando desta forma o paralelismo da execução da aplicação e conseqüente redução no seu tempo de execução. Os processadores usados por uma determinada aplicação constituem a *plataforma de execução* da aplicação.

Plataformas de execução de aplicações paralelas variam em diversos aspectos, dos quais destacamos conectividade, heterogeneidade, compartilhamento, imagem do sistema e escala. *Conectividade* diz respeito aos canais de comunicação que interligam os processadores que compõem a plataforma de execução. Atributos que definem a conectividade de uma plataforma são a topologia, largura de banda, latência e compartilhamento. *Heterogeneidade* trata das diferenças entre os processadores, que podem ser de velocidade e/ou arquitetura. *Compartilhamento* versa sobre a possibilidade dos recursos usados por uma aplicação serem compartilhados por outras aplicações. *Imagem do sistema* se refere à existência de uma visão única da plataforma, independente do processador sendo utilizado. Por exemplo, todos os processadores da plataforma enxergam o mesmo sistema de arquivos? *Escala* estabelece quantos processadores tem a plataforma.

Entender as diferenças entre plataformas é fundamental porque cada aplicação paralela tem uma série de requisitos, que podem ser melhor ou pior atendidos por uma dada plataforma. Em princípio, procuramos executar uma aplicação paralela em uma plataforma adequada às características da aplicação. Por exemplo, considere conectividade, um aspecto em que plataformas diferem consideravelmente. Obviamente, para obter boa performance de uma aplicação paralela cujas tarefas se comunicam e sincronizam freqüentemente, necessitamos utilizar uma plataforma de execução com boa conectividade.

Podemos agrupar as plataformas de execução hoje existentes em quatro grandes grupos: SMPs, MPPs, NOWs e Grids. *SMPs* (ou multiprocessadores simétricos) são máquinas em que vários processadores compartilham a mesma memória. Multiprocessadores possibilitam um fortíssimo acoplamento entre os processadores e executam uma única cópia do sistema operacional para todos os processadores. Portanto, eles apresentam uma imagem única do sistema e excelente conectividade. Todavia, multiprocessadores apresentam limitações em escalabilidade, raramente ultrapassando 16 processadores. Multiprocessadores são relativamente comuns no mercado e vão desde máquinas bipro-

cessadas Intel até grandes servidores como os IBM pSeries. A Figura 2 ilustra a arquitetura de um multiprocessador.

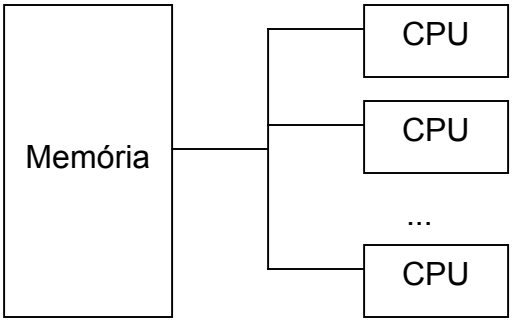

Figura 2 – Multiprocessador

MPPs (ou processadores maciçamente paralelos) são compostos por vários nós (processador e memória) independentes, interconectados por redes dedicadas e muito rápidas. MPPs incluem supercomputadores paralelos como o IBM SP2 e Cray T3E, como também clusters de menor porte montados pelo próprio usuário. Tipicamente cada nó roda sua própria cópia do sistema operacional, mas uma imagem comum do sistema é implementada através da visibilidade dos mesmos sistemas de arquivo por todos os nós. O MPP é controlado por um escalonador que determina quais aplicações executarão em quais nós. Ou seja, não se pode utilizar um nó que não tenha sido alocado à aplicação pelo escalonador. Isto possibilita dedicar partições (um conjunto de nós) às aplicações, permitindo que estas não precisem considerar compartilhamento. Mas, uma vez que aplicações executam em partições dedicadas, pode acontecer que não haja nós suficientes para executar uma aplicação assim que ela é submetida. Neste caso, a aplicação espera em uma fila até que os recursos que solicitou estejam disponíveis. A Figura 3 exemplifica a arquitetura de um MPP.

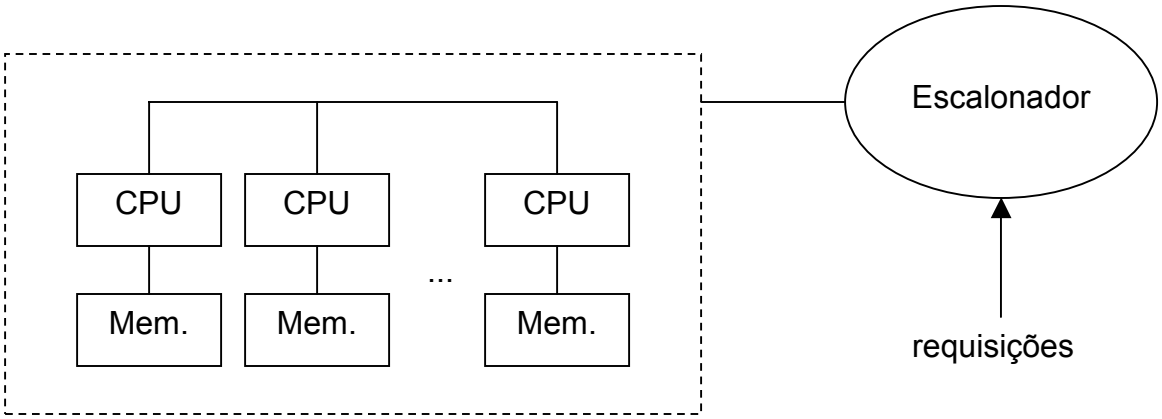

Figura 3 – MPP

NOWs (ou redes de estações de trabalho) são simplesmente um conjunto de estações de trabalho ou PCs, ligados por uma rede local. NOWs são arquiteturalmente semelhantes aos MPPs. Ambas plataformas são formadas por nós que agregam processador e memória. Uma diferença entre NOWs e MPPs é que os nós que compõem uma MPP tipicamente são conectados por redes mais rápidas que as que conectam os nós de NOWs. Mas a principal diferença entre ambas arquiteturas é que NOWs não são escalonadas de forma centralizada. Isto é, em uma NOW, não há um escalonador para o sistema como um todo. Cada nó tem seu próprio escalonador local. Como resultado, não há como dedicar uma partição da NOW a uma só aplicação paralela. Portanto, uma aplicação que executa sobre a NOW deve considerar o impacto sobre sua performance da concorrência por recursos por parte de outras aplicações. A Figura 4 mostra esquematicamente uma NOW.

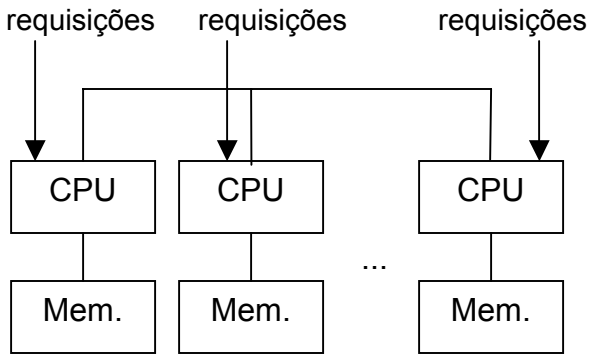

Figura 4 – NOW

Grids são o passo natural depois dos NOWs no sentido de mais heterogeneidade e maior distribuição. Os componentes de um Grid não se restringem a processadores, podendo ser SMPs e MPPs como também instrumentos digitais. Grids tipicamente não fornecem uma imagem comum do sistema para seus usuários. Componentes do Grid podem variar drasticamente em capacidade, software instalado, sistemas de arquivo montados e periféricos instalados. Além disso, os componentes de um Grid podem estar sobre controle de diferentes entidades e, portanto, em domínios administrativos diversos. Conseqüentemente, um dado usuário pode ter acesso e permissões bastante diversas nos diferentes componentes de um Grid. Obviamente, o Grid não pode ser dedicado a um usuário, embora seja possível que algum componente possa ser dedicado (um MPP, por exemplo). É importante salientar que uma aplicação Grid deve estar preparada para lidar com todo este dinamismo e variabilidade da plataforma de execução, adaptando-se ao cenário que se apresenta com o intuito de obter a melhor performance possível no mo-

mento. A Figura 5 exemplifica um possível Grid, composto por um MPP e computadores de vários tipos conectados via Internet. Note que um destes computadores realiza instrumentação (no exemplo, através de um microscópio), enquanto outro computador dispõe de grande capacidade para armazenamento de dados.

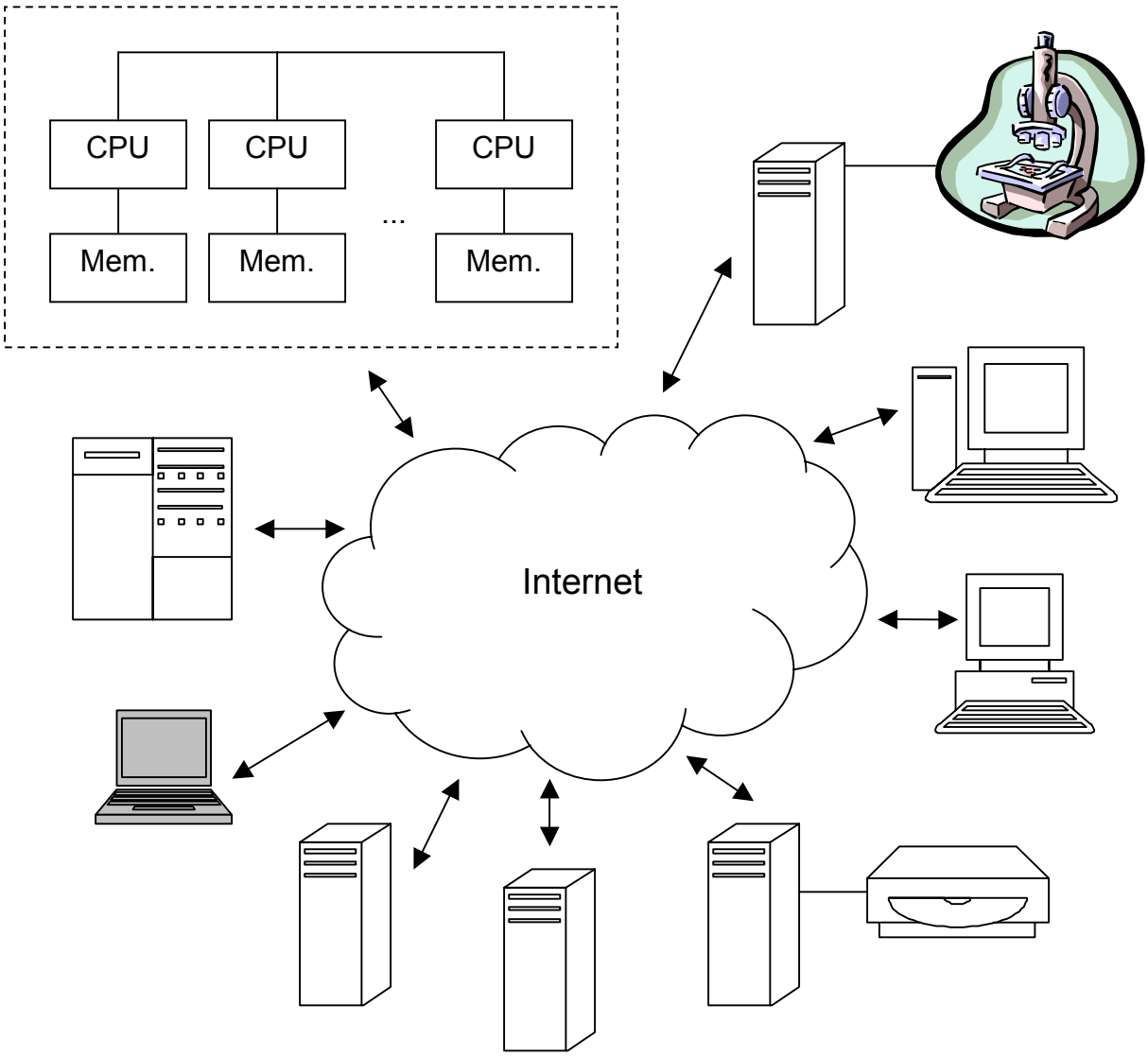

Figura 5 – Grid Computacional

A Tabela 1 sumariza as características das plataformas de execução de aplicações paralelas discutidas aqui. Mantenha em mente, entretanto, que a Tabela 1 descreve o comportamento “médio” ou “típico” dos diferentes tipos de plataformas de execução. Certas plataformas podem apresentar características arquiteturais adicionais que impactam na performance das aplicações paralelas que nela executam. Por exemplo, alguns MPPs oferecem suporte de hardware a memória compartilhada, através de uma tecnologia de-

nominada DSM (Distributed Shared Memory), o que melhora o desempenho de aplicações baseadas em memória compartilhada. Uma vez que Grids são o nosso foco neste texto, referimos [33] caso o leitor queira mais detalhes sobre plataformas de execução de aplicações paralelas tradicionais (SMPs, MPPs e NOWs).

	SMPs	**MPPs**	**NOWs**	**Grids**
Conectividade	excelente	muito boa	boa	média/ruim
Heterogeneidade	nula	baixa	média	alta
Compartilhado	não	não	sim	sim
Imagem	única	comum	comum	múltipla
Escala	10	1.000	1.000	100.000

Tabela 1 – Resumo das características típicas de diferentes plataformas de execução

Mesmo quando não há distinções arquiteturais, diferentes plataformas do mesmo tipo podem diferir consideravelmente. Em particular, um Grid pode diferir radicalmente de outro. Por exemplo, considere o TeraGrid [37] e o SETI@home [34]. O TeraGrid é um Grid que interliga 4 centros de supercomputação norte-americanos através de canais de altíssima velocidade (40 GigaBits/segundo). Cada um dos 4 centros terá milhares de processadores *dedicados* ao TeraGrid, gerando um poder agregado de 13,6 TeraFlops. O SETI@home, por outro lado, utiliza a capacidade computacional ociosa de computadores que se juntam voluntariamente ao sistema através da instalação do software cliente do projeto. Em fevereiro de 2000, SETI@home contava com 1.6 milhões de processadores espalhados em 224 países, e computava em média a uma velocidade de 10 Teraflops Embora o SETI@home tenha reportado uma performance compatível com o TeraGrid, é patente a diferença entre os dois Grids. O TeraGrid é muito mais *acoplado* que o SETI@home.

O conceito de *acoplamento do Grid* (i.e. quão próximos estão seus componentes) é fundamental para compreendermos quais aplicações podem executar eficientemente em um Grid. Voltando ao exemplo acima, o SETI@home se presta somente para execução de *aplicações leves* (i.e. levemente acopladas), enquanto o TeraGrid pode propiciar condições para execução eficiente de *aplicações pesadas* (i.e. fortemente acopladas).

3 Aspectos da Computação em Grid

Grids levantam as mesmas questões que outras plataformas de execução de aplicações paralelas, tais como modelo de programação (troca de mensagens $\times$ memória compartilhada) e balanceamento de carga. Além destas questões, entretanto, Grids computacionais trazem uma nova gama de aspectos que precisam ser considerados pelos desenvolvedores de infraestrutura e de aplicações. Estes novos aspectos surgem devido às próprias características do Grid, como ampla distribuição, grande escala, alta heterogeneidade e múltiplos domínios administrativos.

3.1 Escalonamento de Aplicação

Tradicionalmente, há um escalonador que controla os recursos do sistema (i.e., não há como usar os recursos sem a autorização do escalonador). Por exemplo, o sistema operacional controla o computador no qual roda, decidindo quando e aonde (no caso de multiprocessadores) cada processo executa. Chamaremos estes escalonadores de *escalonadores de recursos*. Uma característica importante dos escalonadores de recurso é que eles recebem solicitações de vários usuários e, portanto, tem que arbitrar entre estes vários usuários o uso dos recursos que controlam.

Devido à grande escala, ampla distribuição e existência de múltiplos domínios administrativos, não é possível construir um escalonador de recursos para Grids. Uma razão para isto é que sistemas distribuídos que dependem de uma visão global coerente (necessária ao *controle* dos recursos) escalam mal. Além disso, é muito difícil (senão impossível) convencer os administradores dos recursos que compõem o Grid a abrir mão do *controle* de seus recursos.

Assim sendo, para utilizar recursos controlados por vários escalonadores de recurso distintos, alguém tem que (i) escolher quais recursos serão utilizados na execução da aplicação, (ii) estabelecer quais tarefas cada um destes recursos realizará, e (iii) submeter solicitações aos escalonadores de recurso apropriados para que estas tarefas sejam executadas. Esta é a tarefa do *escalonador de aplicação*. Escalonadores de aplicação não controlam os recursos que usam. Eles obtêm acesso a tais recursos submetendo solicitações para os escalonadores que controlam os recursos.

Ou seja, em um Grid, as decisões de escalonamento são divididas em duas camadas, com parte da responsabilidade pelo escalonamento sendo transferida dos escalonadores de recurso para o nível de aplicação. A Figura 6 ilustra o escalonamento em um Grid Computacional.

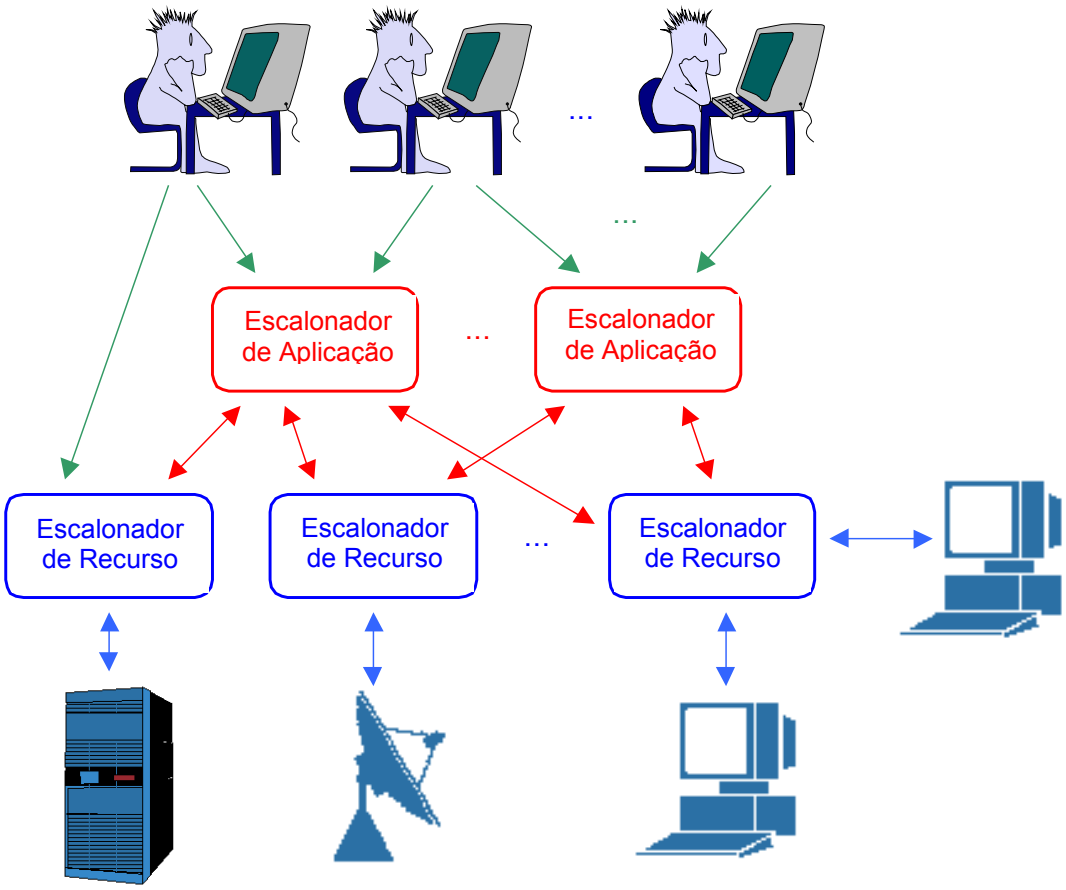

Figura 6 – Escalonamento em um Grid Computacional

As decisões tomadas pelo escalonador de aplicações (quais recursos serão utilizados e quais tarefas cada um destes recursos realizará) são normalmente baseadas em um modelo do desempenho da aplicação e em dados sobre o estado atual dos vários recursos que formam o Grid [4] [6] [35] [40] [41] [43]. Portanto, escalonadores de aplicação têm que conhecer detalhes das aplicações que escalonam, i.e. eles são construídos com uma aplicação (ou classe de aplicações) em mente. Além disso, escalonadores de aplicação normalmente precisam saber quanto tempo cada recurso vai levar para processar uma dada tarefa. Sem esta informação, é difícil escolher os melhores recursos a utilizar, como também determinar qual tarefa alocar a cada um desses recursos. Para obter tal informação, foram desenvolvidos sistemas que monitoram e prevêem o comportamento futuro de diversos tipos de recursos [29] [42]. Este esquema de obtenção de informação baseado em monitoração tem se mostrado eficaz quando os recursos monitorados são redes TCP/IP ou computadores compartilhados no tempo, mas ainda há questões quanto a escalabilidade dos sistemas de monitoração [21].

Uma questão pela arquitetura de escalonamento para Grids (ilustrada pela Figura 6) é a determinação do *comportamento emergente* do escalonadores de aplicação. Ou seja, qual o impacto no sistema como um todo de ter vários escalonadores de aplicação, cada qual tentando obter boa performance para sua aplicação. Essa é uma questão bastante relevante pois há evidência teórica que sistemas nos quais alocação de recursos é feita por várias entidades independentes podem exibir degradação de performance [30] ou até comportamento caótico [26].

Jacobi AppLeS

Jacobi AppLeS [4] é um exemplo interessante por ter lançado a idéia de escalonamento de aplicações e também por escalonar uma aplicação bem conhecida (Jacobi). Jacobi é um método usado para resolver a aproximação por diferenças finitas da equação de Poisson e portanto é aplicável a problemas que envolvem fluxo de calor, eletrostática e gravitação. Além de ser interessante por si só, Jacobi pode ser visto como uma instância de uma importante classe de aplicações paralelas: aplicações fortemente acopladas de paralelismo em dados.

Jacobi AppLeS é um escalonador para Jacobi 2D. Em Jacobi 2D, o domínio do problema é discretizado em uma matriz bidimensional. Em cada iteração, cada elemento da matriz é atualizado com a média dos seus quatro vizinhos. Jacobi termina por convergência, isto é, quando uma iteração altera muito pouco os elementos da matriz.

Quando Jacobi é executado em um MPP, a matriz bidimensional é tipicamente dividida em ambas as dimensões, gerando submatrizes de igual tamanho. Cada submatriz é então alocada a um processador. A cada iteração, portanto, é necessário comunicação entre processadores para troca das fronteiras das submatrizes. A Figura 7 mostra a distribuição de dados entre 4 processadores de um MPP alocados para executar Jacobi. Como, em um MPP, os processadores são idênticos e dedicados e a comunicação é muito boa entre quaisquer dois processadores, esta simples estratégia de alocação de trabalho balanceia a carga entre os processadores, garantindo bom desempenho.

Em um Grid, entretanto, processadores e canais de comunicação são heterogêneos. Além disso, outras aplicações estão concorrendo pelos mesmos recursos (processadores e canais de comunicação) enquanto Jacobi executa. Conseqüentemente, a estratégia descrita acima provavelmente vai produzir um desbalanço de carga, afetando o desempenho. Mais ainda, uma vez que as condições de carga do Grid variam dinamicamente, o que é uma boa divisão de carga vai variar a cada execução da aplicação. Finalmente, devido à existência de canais de comunicação mais lentos e compartilhados

com outras aplicações, talvez não valha a pena utilizar todos os processadores disponíveis.

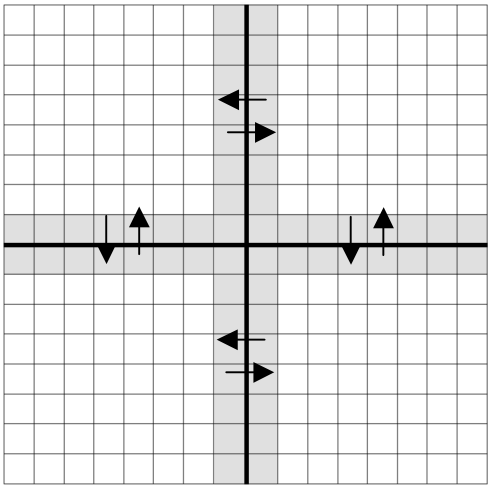

Figura 7 – Jacobi rodando com 4 processadores em um MPP

A solução oferecida por AppLes Jacobi se baseia em três elementos principais. Primeiro, o escalonamento em si é simplificado pela decisão de utilizar um particionamento unidimensional. Segundo, o escalonador se utiliza do NWS [42] para obter *previsões* de curto prazo da disponibilidade de cada processador e da latência e banda da comunicação entre quaisquer dois processadores. Terceiro, o escalonador dispõe de um *modelo de performance* da aplicação, que é usado para avaliar suas decisões. Este modelo é o seguinte:

$T_i = A_i \times P_i + C_i$, onde:

T_i é o tempo para o processador *i* executar uma iteração

A_i é a área da submatriz alocada ao processador *i*

P_i é o tempo que o processador *i* leva para computar um elemento

C_i é o tempo que o processador *i* leva para comunicar suas fronteiras

Note que P_i e C_i são estimados com base nos dados fornecidos pelo NWS.

O escalonamento propriamente dito começa ordenando os processadores por uma "distância" específica da aplicação (que cresce quadraticamente com a diferença de velocidade dos processadores e linearmente com a diferença de suas capacidades de comunicação). Uma vez os processadores ordenados, tenta-se iterativamente uma solução com os *n* primeiros processadores, até que a solução com *n* - 1 processadores se mostre mais rápida, ou até que não haja mais processadores. Naturalmente, o tempo de uma

iteração é estimado como o maior T_i de todos os processadores. Fixados n processadores, a solução de escalonamento é obtida dividindo a matriz proporcionalmente a P_i.

Por exemplo, suponha que o Grid tem quatro processadores: P_0, P_1, P_2 e P_3. Assuma ainda que P_0 e P_1 tem o dobro da velocidade de P_2 e P_3, que P_1 tem uma outra aplicação rodando e só poderá dedicar 50% de seu poder computacional a aplicação, que P_3 está conectado a uma rede que vivencia intenso trafego e que sua comunicação está ordens de grandeza mais lenta que entre os demais processadores. Uma vez que P_3 está se comunicando muito lentamente, o AppLeS não vai utilizá-lo para esta execução. Note que esta decisão não descarta a possibilidade que P_3 venha a ser usado em uma futura execução da aplicação, quando as condições da rede forem diferentes. Note também que, embora P_1 seja duas vezes mais rápido que P_2, uma vez que só 50% de P_1 está disponível, P_1 e P_2 são idênticos para a aplicação (pelo menos nesta execução). A Figura 8 mostra o resultado que o AppLeS Jacobi produziria neste cenário.

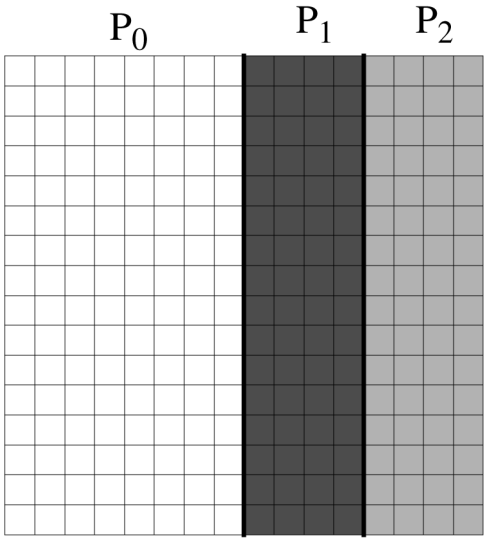

Figura 8 – Escalonado feito pelo AppLeS Jacobi [4]

Devemos salientar que um aspecto importante para o bom funcionamento do Jacobi AppLeS é o tempo relativamente curto de execução da aplicação. O tempo de execução dos experimentos descritos em [4] é da ordem de segundos, casando perfeitamente com as previsões de curto prazo do NWS. Para aplicações que executam por mais tempo (horas, digamos), seria necessário prever a disponibilidade de recursos do Grid por prazos mais longos. Uma alternativa interessante seria construir um escalonador que, além do escalonamento inicial, continuasse funcionando para *reescalonar a aplicação* caso as condições do Grid mudassem consideravelmente [35]. Neste caso, naturalmente,

a aplicação teria que ser escrita para permitir tal reescalonamento, suportando a redistribuição de trabalho durante a execução.

Work Queue with Replication (WQR)

Note que as previsões de performance fornecidas pelo NWS são fundamentais para o funcionamento do Jacobi AppLeS. De fato, a vasta maioria dos escalonadores de aplicação descritos na literatura [4] [6] [35] [40] [41] [43] utiliza alguma forma de previsão de performance. Infelizmente, há problemas em aplicar em larga escala os sistemas de monitoração e previsão existentes (como NWS [42] e Remos [29]), especialmente quando se trata de prever o comportamento dos canais de comunicação, que crescem quadraticamente com o número de máquinas do Grid. Veja [21] para ter uma idéia da dificuldade de se construir um sistema escalável para monitoramento de canais de comunicação.

Como parte do projeto MyGrid [8] [9], nós desenvolvemos o Work Queue with Replication (WQR) [32], um escalonador capaz de obter boa performance para a aplicação no ambiente muito dinâmico que é o Grid sem depender de previsões de performance dos componentes do Grid. Isto é possível devido a dois fatores fundamentais. Primeiro, WQR usa replicação de tarefas para garantir a boa performance da aplicação. A idéia é gastar alguns ciclos a mais para compensar pela falta de informação sobre o ambiente. Segundo, WQR escalona aplicações relativamente simples. Mais precisamente, WQR escalona aplicações Bag of Tasks. Aplicações Bag of Tasks são aquelas cujas tarefas são independentes, i.e. não se comunicam e podem ser executadas em qualquer ordem. Aplicações Bag of Tasks são importantes porque (i) são usadas por várias áreas, tais como mineração de dados, pesquisas massivas (como quebra de chave), varredura de parâmetros, simulações Monte Carlo, fractais e manipulação de imagens (como tomografia), e (ii) são bastante apropriadas para execução no Grid, devido exatamente a sua fraca acoplamento (veja seção 2).

O algoritmo do WQR é bastante simples. Uma fila de tarefas é criada na submissão da aplicação. Sempre que há um processador disponível, uma tarefa é enviada para este processador. Quando não há mais tarefas para enviar (i.e. a fila de tarefas está vazia), uma das tarefas em execução é replicada. Quando uma das replicas termina, as demais replicas são abortadas pelo escalonador. Para evitar o desperdício de poder computacional, estabelecemos o máximo de replicas que uma tarefa pode ter. Felizmente, nossos experimentos indicam que grande parte do ganho de performance obtido pelo WQR se manifesta com apenas 2 replicações.

De fato, considere a Figura 9, que mostra a performance do WQR em comparação com o WQ (Work Queue, que funciona sem replicação), o Sufferage (um bom escalona-

dor baseado em informações sobre o Grid e sobre as tarefas), e com o Dynamic FPLTF (Dynamic Fastest Processor to Largest Task First, outro bom escalonador que utiliza informações sobre o Grid e sobre as tarefas). A Figura 9 apresenta o tempo médio obtido pelos quatro algoritmos de escalonamento em função da heterogeneidade das tarefas (quanto maior, mais heterogêneo). Note que WQR foi executado três vezes, com replicação máxima de 2, 3 e 4 processadores. Observe também que Sufferage e Dynamic FPLTF tiveram *informação perfeita* a respeito das tarefas e do Grid, algo inatingível na prática. Portanto, é um excelente resultado o fato de WQR teve desempenho comparável com Sufferage e Dynamic FPLTF baseados em informação perfeita.

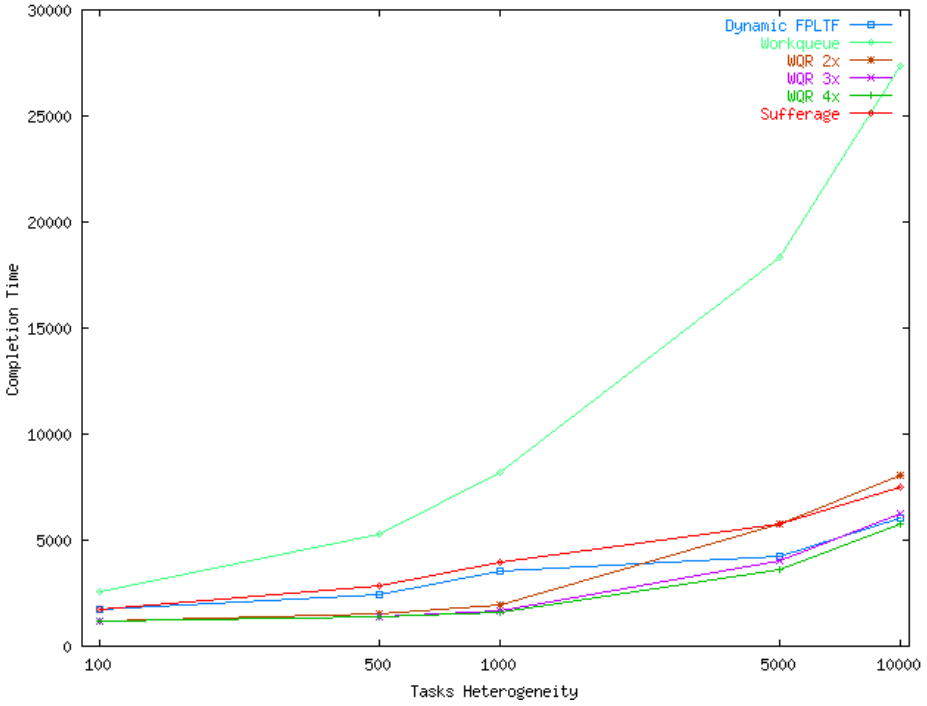

Figura 9 – Desempenho do WQR

Obviamente, WQR consome mais ciclos que os algoritmos de escalonamento tradicionais. A Figura 10 mostra o consumo adicional de CPU em função da heterogeneidade das tarefas. Em situações que a heterogeneidade de tarefas é pequena, este consumo não é significativo, não ultrapassando 15%. Por outro lado, quando tarefas são muito heterogêneas, WQR desperdiça uma quantidade considerável de ciclos. De fato, o desperdício pode chegar próximo a 100%. Entretanto, tal problema pode ser controlado pela limitação do número máximo de replicas de uma tarefa. Quando limitamos WQR a usar 2 replicas (WQR 2x), temos que o desperdício de CPU fica sempre abaixo de 40%.

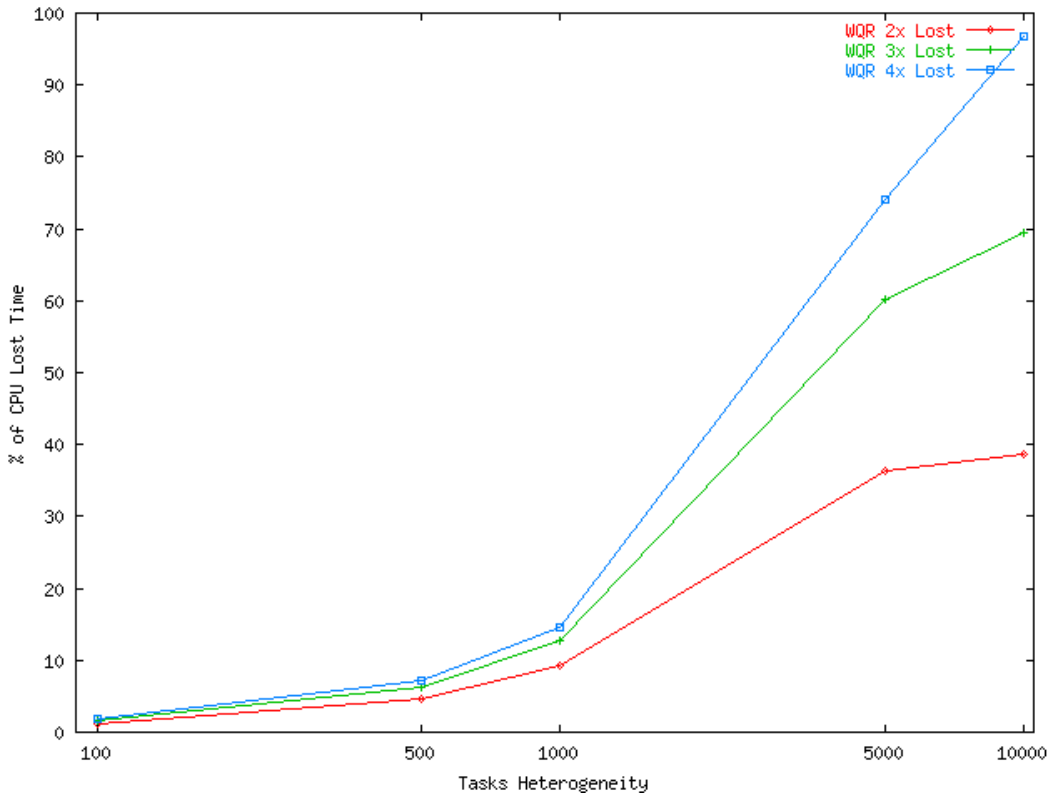

Figura 10 – Desperdício de ciclos do WQR

3.2 Acesso e Autenticação

Devido à ampla distribuição e existência de múltiplos domínios administrativos, acesso aos recursos que compõem um Grid não é tão simples quanto em outras plataformas de execução de aplicações paralelas. Por exemplo, um SMP roda uma só cópia do sistema operacional. Ao efetuar login no sistema, portanto, o usuário é identificado e autenticado para todos os processadores. Da mesma forma, embora tenham várias cópias do sistema operacional, MPPs e NOWs normalmente mantêm um cadastro de usuários que é válido para todos os processadores do sistema.

Já em Grids, é necessária uma forma de acesso para cada recurso (ou subconjunto de recursos, quando estes compartilham do mesmo cadastro de usuários) que compõe o Grid. Obviamente, tal forma de acesso tem que oferecer garantias sobre autenticação dos usuários, caso contrario os administradores de sistema se oporão a seu uso.

Globus GRAM

Como discutiremos em detalhes na seção 4.1, a solução Globus para Computação Grid é composta por vários serviços distintos, porém integrados. GRAM (Globus Resource Allocation Manager) é um destes componentes. GRAM fornece uma interface

uniforme para submissão e controle de tarefas [11], escondendo a multiplicidade de escalonadores de recursos dos demais serviços de Grid (do escalonador de aplicações, por exemplo). Além disso, GRAM informa sobre o status do recurso ao MDS (o serviço Globus que fornece informação sobre o Grid).

Um exame da arquitetura do GRAM (mostrada na Figura 11) esclarece bastante sobre seus objetivos e funcionamento. A Figura 11 mostra os três componentes do GRAM (Gatekeeper, Job Manager e GRAM Reporter), bem como componentes externos que interagem com o GRAM. O cliente GRAM é aquele que o utiliza para submeter e controlar a execução de tarefas. Note que o cliente GRAM pode ser um escalonador de aplicação ou até o próprio usuário. Para o cliente, a grande vantagem de usar GRAM é a manipulação *uniforme* de tarefas, i.e. a submissão e controle de tarefas não importando qual é o escalonador de recurso (Local Resource Manager, na Figura 11) usado para controlar a máquina. Isto é possível porque as requisições enviadas ao GRAM são sempre escritas em RSL (Resource Specification Language), independentemente de qual escalonador de recurso esteja sendo utilizado. O Job Manager é o responsável por converter a requisição em RSL em um formato que o escalonador de recurso em questão entenda. Ao que sabemos, há versões do Job Manager que interfaceiam com Condor, NQE, Codine, EASY, LSF, LoadLeveler, PBS, Unix e Windows.

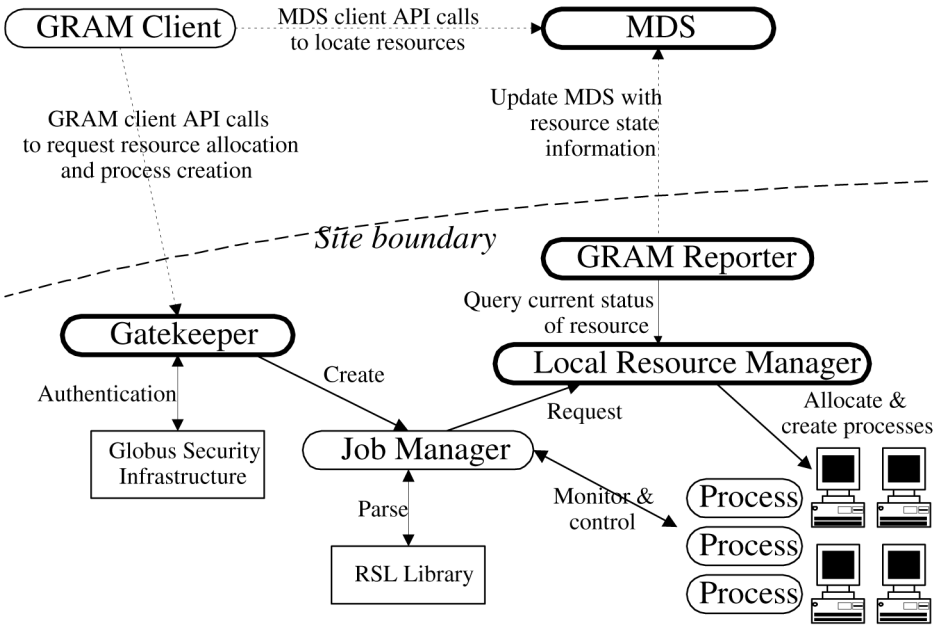

Figura 11 – Arquitetura do GRAM [11]

As requisições enviadas pelo cliente são recebidas pelo Gatekeeper, que consulta o Globus Security Infrastructure (GSI) para identificar o usuário e verificar se ele pode executar na máquina em questão (veja a seção 4.1 para maiores detalhes). Caso o usuário tenha permissão, uma submissão de tarefa cria um Job Manager que é responsável por iniciar e monitorar a tarefa. Requisições sobre o estado da tarefa serão encaminhadas diretamente ao Job Manager.

Por fim, o GRAM Reporter obtém informações de status e carga da máquina junto ao escalonador de recursos (Local Resource Manager, na Figura 11) e as repassa para o MDS. O MDS, por sua vez, torna estas informações disponíveis sob demanda para os outros componentes da arquitetura Globus. Por exemplo, um escalonador de aplicação pode se basear nas informações fornecidas pelo MDS para decidir quais recursos utilizar na execução de uma dada aplicação.

3.3 Economias Grids

Para que Grids sejam úteis, é preciso que eles existam, é preciso criá-los. Esta sentença é bastante óbvia. Contudo, ela levanta alguns problemas técnicos não-triviais. Suponha que duas instituições *A* e *B* decidem formar um Grid, unificando seus recursos e fornecendo melhores serviços computacionais a seus usuários. Mas, como os recursos serão compartilhados? Suponha que *A* tem mais que o dobro dos recursos de *B*. Um compartilhamento equânime seria prejudicial a *A*, que então relutaria em formar um Grid com *B*. Por outro lado, assuma que *A* não pode fornecer acesso a seus recursos durante o expediente bancário (10:00 as 16:00). Como se pode perceber, o *contrato* entre *A* e *B* para compartilhamento de recursos e construção de um Grid comum pode ser algo bastante sofisticado. Tal sofisticação gera uma pergunta óbvia de como as regras de compartilhamento acordadas serão implementadas e policiadas.

Se a criação de um Grid entre *duas* instituições pode oferecer tal complexidade, imagine a criação de Grids envolvendo centenas ou milhares de entidades. A abordagem que vem sendo sugerida para este problema é a criação de Economias Grid [5] [7]. A idéia básica é a construção de um mercado computacional onde as diversas entidades envolvidas no Grid possam trocar recursos. O aspecto mais atraente desta abordagem é que acordos de compartilhamento sofisticados não são mais necessários. Recursos são comprados e vendidos de forma independente, sem um supervisor onisciente do mercado. Desta forma, entidades podem decidir independentemente quando comprar ou vender recursos. A moeda utilizada seria provavelmente algum “dinheiro virtual” que daria apenas poder de compra de recursos computacionais. Entretanto, é concebível o câmbio

entre este “dinheiro virtual” e dinheiro real, incentivando a criação de empresas que forneçam poder computacional sobre demanda.

Computational Co-op

Computacional Co-op tem por objetivo possibilitar que sites independentes se juntem em um Grid [7]. Um site pode ser uma MPP ou uma NOW. Uma vez que os sites são independentes, a adesão ao Grid tem que mutuamente beneficiar todos os sites envolvidos. A necessidade de mútuo beneficio para criação do Grid gera a analogia entre o Grid e uma Cooperativa, dando assim nome ao projeto. É importante também salientar que o Co-op assume que as aplicações sejam do tipo mestre-escravo.

Para viabilizar que sites independentes se federem em um Grid, o Co-op introduz um mecanismo que permite (i) controlar quanto dos recursos locais estão sendo destinados para o Grid, e (ii) garantir a quantidade de recursos que o site está recebendo do Grid. A idéia é que a combinação deste controle e desta garantia possibilite que sites independentes façam acordos de cooperação.

O mecanismo que viabiliza tanto o controle da contribuição local como a garantia da contrapartida é o escalonamento proporcional (proportional-share scheduling). No escalonamento proporcional, os recursos são controlados por tickets. Cada usuário (ou entidade que pode usar o sistema) recebe um determinado número de tickets. Quando da submissão de uma aplicação, o usuário “empresta” um determinado número de tickets a aplicação. Seja T a soma dos tickets das aplicações em execução e t_i os tickets alocados a aplicação i. Escalonamento proporcional significa que a aplicação i vai receber t_i / T dos recursos disponíveis. O Co-op faz um contribuição interessante a literatura de escalonamento ao estender o Stride Scheduler [37] para controlar múltiplos processadores não-dedicados, o que viabiliza o uso do Co-op em NOWs.

Cada site, então, executa o escalonador proporcional introduzido pelo Co-op. Parte dos tickets de cada site são destinados ao Grid, permitindo assim que os sites controlem quantos recursos locais são alocados ao Grid. Os tickets que os sites destinam ao Co-op servem como lastro para criação de novos tickets: os tickets do Co-op. Os tickets do Co-op são então divididos entre os sites, fornecendo a cada site uma garantia dos recursos do Grid que estarão disponíveis para o site. O esquema se completa pelo câmbio de tickets do Co-op para tickets do site remoto quando se submete uma tarefa para um site remoto. Como era de se esperar, este câmbio é também proporcional.

Julgamos o Co-op um trabalho interessante e até pioneiro. Entretanto, nossas tentativas de aplicá-lo na prática revelaram que a alocação de recursos proporcional é por demais rígida. Aparentemente, os sites participantes de um Grid desejam muito mais

flexibilidade para estabelecer o que contribuem com o Grid e definir o que esperam de volta. O Co-op infelizmente não fornece esta flexibilidade. Parte do problema parece ser que tickets não são “gastáveis” e portanto a alocação negociada é fixa. Soluções baseadas na criação de mercados para compra e venda de recursos podem portanto fornecer uma solução interessante para esta questão.

3.4 Imagem do Sistema

Ao usamos um computador, dependemos das abstrações criadas pelo sistema operacional, tais como arquivos, diretórios, permissões e processos, para lidarmos com o sistema. São tais abstrações que nos permitem expressar o queremos fazer. Elas também nos permitem nomear os dados persistentes que temos armazenados no sistema. Através destas abstrações básicas fornecidas pelo sistema operacional, o usuário tem uma *imagem do sistema*, formada pelo conjunto de objetos que ele pode manipular e pelas regras de manipulação destes objetos.

Plataformas de execução de aplicações paralelas que tem uma única instância do sistema operacional (SMPs) automaticamente fornecem a seus usuários uma imagem única do sistema. Já em plataformas que contém várias instâncias do sistema operacional (MPPs, NOWs e Grids), é necessário *construir* uma imagem consistente do sistema. Uma imagem consistente do sistema cria a ilusão (ainda que imperfeita) que os objetos que o usuário pode manipular são acessíveis da mesma forma de qualquer processador que compõe a plataforma.

MPPs e NOWs contam com boa conectividade e administração centralizada. Isso permite a configuração dos processadores que compõem a plataforma para compartilhar o mesmo cadastro de usuários e os sistemas de arquivo mais importante (o `/home`, por exemplo), criando assim uma imagem razoavelmente consistente do sistema. Grids, por outro lado, são amplamente dispersos e muitas vezes sob controle de diversas entidades administrativas distintas. Não é factível, por exemplo, simplesmente montar o mesmo `/home` em todos os processadores que compõem o Grid, pois, além de problemas de desempenho, há também questões administrativas. Por outro lado, não queremos deixar que o usuário tenha que lidar com várias imagens totalmente distintas do sistema.

As soluções para este problema dividem-se em dois grandes grupos, aquelas que evitam os problemas administrativos trabalhando à nível de usuário [28] [38] e aquelas que introduzem novas abstrações para que o usuário possa lidar com o Grid [8] [9]. Em princípio, soluções à nível de usuário são mais simples de usar pois suportam abstrações já conhecidas pelo usuário (e.g. arquivos). Entretanto, elas podem apresentar sérios pro-

blemas de performance dependendo da aplicação e da conectividade do Grid. Novas abstrações, ao contrário, requerem o aprendizado de novos conceitos, mas podem vir a oferecer uma forma mais eficiente de usar o Grid.

Condor

Condor é uma sistema que objetiva fornecer grande quantidade de poder computacional a médio e longo prazo (dias a semanas) utilizando processadores ociosos na rede [28]. O usuário submete a Condor várias tarefas independentes para execução. Condor localiza máquinas que estejam ociosas e envia tais tarefas para nelas executarem. Mais detalhes sobre o sistema Condor serão apresentados na seção 4.2. Por agora, nosso interesse é como Condor resolve a questão da imagem do sistema.

O usuário Condor tem a ilusão que as tarefas submetidas executam localmente em sua máquina base (i.e. o computador usado pelo usuário). Para propiciar o acesso a maior quantidade possível de recursos, Condor não assume que a máquina ociosa usada para execução da tarefa monta os mesmos sistemas de arquivo que a máquina base, na qual o usuário submeteu a tarefa. Condor soluciona o problema redirecionando chamadas ao sistema operacional efetuadas na máquina remota para a máquina base do usuário, como exemplificado pela Figura 12.

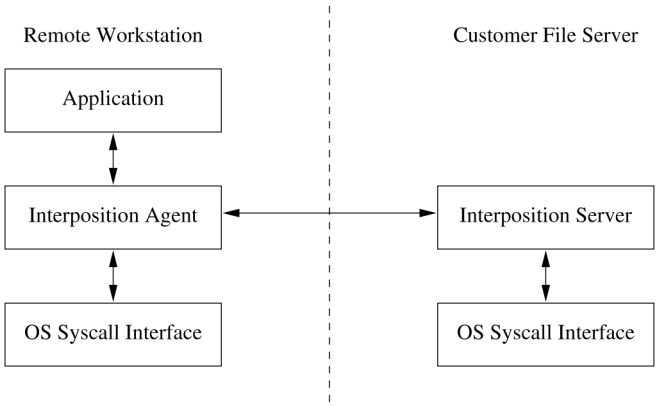

Figura 12 – Redirecionamento de chamadas ao sistema operacional usado por Condor [3]

Tal redirecionamento é implementado através da biblioteca de redirecionamento Condor. A biblioteca de redirecionamento Condor oferece as mesmas rotinas de manipulação de arquivo que a biblioteca do sistema (i.e. `open`(), `read`(), `write`(), etc). Uma vez interceptada a chama ao sistema de arquivo, biblioteca de redirecionamento Condor decide se a operação solicitada deve ser realizada localmente (arquivo temporário, por exemplo) ou na máquina base do usuário (arquivo em `/home`, por exemplo). Caso a opera-

ção deva ser local, o sistema operacional é invocado. Caso a operação deva ser realizada na máquina base, um RPC é feito para um servidor de requisições que o Condor deixa rodando na máquina base do usuário.

Note que o esquema de redirecionamento utilizado por Condor implica que a tarefa a ser submetida pelo usuário ao sistema deve ser link-editada de forma especial, i.e. a tarefa deve ser link-editada com a biblioteca Condor, não com a biblioteca padrão do sistema. (Vale mencionar que Condor também executa tarefas link-editadas com a biblioteca do sistema. Neste caso, obviamente, o redirecionamento não funciona, limitando assim onde tal tarefa pode executar ou deixando para o usuário a responsabilidade de lidar com múltiplas imagens do sistema.)

4 Sistemas para Computação em Grid

Vários sistemas para suporte à Computação em Grid surgiram nos últimos anos, tanto através de esforços acadêmicos (e.g. Globus, Legion, Condor, MyGrid), quando decorrentes de empreendimento comerciais (e.g. Entropia, distributed.net). Dentre os esforços acadêmicos, Globus foi de longe o projeto que teve maior impacto. Todavia, Globus não soluciona todos os problemas existentes na Computação em Grid. É importante também conhecer alternativas e, principalmente, soluções complementares a Globus.

Quanto aos esforços comerciais, ainda é muito cedo para determinar seu impacto. Além disso, pela própria natureza comercial destes esforços, muito menos detalhes técnicos estão disponíveis sobre o funcionamento de tais sistemas. Em particular, um aspecto que necessita melhor definição por parte da Entropia e da distributed.net diz respeito a *abertura* dos seus sistemas, i.e. a capacidade de interoperar com outros sistemas para Computação em Grid.

4.1 Globus

Globus consiste de um conjunto de serviços que facilitam Computação em Grid [16]. Os serviços Globus podem ser usados para submissão e controle de aplicações, descoberta de recursos, movimentação de dados e segurança no Grid. Os principais serviços Globus disponíveis atualmente (na versão 2.0) são:

Serviço	**Funcionalidade**
GSI	Segurança, autenticação única no Grid
GRAM	Submissão e controle de tarefas
Nexus	Comunicação entre tarefas
MPI-G	MPI sobre Nexus
MDS	Informações e diretórios
GASS	Transferência de arquivos
GridFTP	Transferência de arquivos

Tabela 2 – Principais serviços Globus

Segurança e Autenticação

Um aspecto que complica o uso de Grids na prática é a autenticação de usuários em diferentes domínios administrativos. Em princípio, o usuário tem que se autenticar para cada domínio administrativo de uma forma determinada pelo administrador do domínio (que tipicamente envolve fornecer uma identificação de usuário e uma senha). Este esquema coloca uma grande carga no usuário (quem usa vários sites Web que exigem login, tem uma idéia bem concreta destes problemas). No contexto de Computação em Grid, os problemas de múltipla autenticação são agravados pois queremos ter programas que possam efetuar ações que exigem autenticação (e.g. submeter uma tarefa a um site remoto).

GSI (Globus Security Infrastructure) é o serviço Globus que ataca este problema. GSI viabiliza o *login único* no Grid. GSI utiliza criptografia de chave pública, certificados X.509 e comunicação SSL (Secure Sockets Layer) para estabelecer a *identidade Globus* do usuário. Por exemplo, "C=US, O=University of California San Diego, OU=Grid Computing Lab, CN=Walfredo Cirne" era nossa identidade em Gusto (o primeiro Grid montado com Globus). Depois do usuário ter se identificado junto ao GSI, todos os demais serviços Globus saberão, de forma segura, que o usuário é de fato quem diz ser.

Uma vez que um serviço sabe a identidade Globus do usuário, resta estabelecer quais operações tal usuário pode realizar. Isto é feito mapeando a identidade Globus para um usuário local. Por exemplo, o serviço GRAM (submissão e controle de tarefas, veja seção 3.2) instalado em `thing1.ucsd.edu` mapeava "C=US, O=University of California San Diego, OU=Grid Computing Lab, CN=Walfredo Cirne" para `walfredo`. Já o GRAM que rodava em `bluehorizon.sdsc.edu` mapeava "C=US, O=University of California San Diego, OU=Grid Computing Lab, CN=Walfredo Cirne" para `u15595`.

Alocação e Descoberta de Recursos

Como discutimos na seção 3.1, Grids não têm um escalonador que controla todo o sistema. Assim sendo, quando um usuário que submeter uma aplicação para execução no Grid, o usuário utiliza um escalonador de aplicação que escolhe os recursos a utilizar, particiona o trabalho entre tais recursos, e envia tarefas para os escalonadores dos recursos, como ilustrado pela Figura 6.

Em Globus, os escalonadores de recurso são acessados através do serviço GRAM (discutido na seção 3.1). GRAM fornece uma interface única que permite submeter, monitorar e controlar tarefas de forma independente do escalonador de recursos. Assim sendo, escalonadores de aplicação não precisam entender dos detalhes particulares de cada

escalonador de recurso. Para facilitar ainda mais a tarefa dos escalonadores de aplicação, Globus também disponibiliza MDS (Metacomputing Directory Service), um serviço de informação sobre o Grid. MDS contém informações sobre os recursos que formam o Grid e também sobre seu estado (carga, disponibilidade, etc).

Uma idéia bastante interessante em Globus é que escalonadores de aplicação podem usar os serviços de outros escalonadores de aplicação. O escalonador que recebe a solicitação do cliente lida com a especificação em mais alto nível. Ele refina tal especificação e, para implementá-la, submete novas solicitações a escalonadores de recurso (que de fato executam solicitações) e/ou escalonadores de aplicação (que utilizam outros escalonadores para executar solicitações).

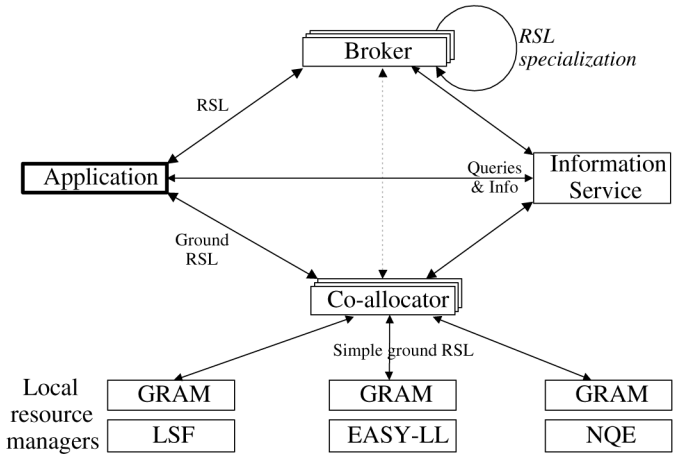

Figura 13 – Processo de especialização de requisição [16]

Globus suporta bem esta hierarquia de escalonadores através da linguagem RSL (Resource Specification Language). RSL é capaz de expressar tanto solicitação de alto nível (como a que o usuário envia a seu escalonador de aplicações), como também solicitações concretas (que são enviadas para GRAMs, que as traduzem para escalonadores de recurso locais). Portanto, o trabalho de um escalonador de aplicação em Globus pode ser descrito como sendo o de refinar solicitações RSL. A Figura 13 ilustra este processo. Note que Globus usa Broker para o que chamamos de escalonador de aplicação. Já o co-alocador (Co-allocator) é um escalonador de aplicação especializado em garantir que tarefas localizadas em máquinas distintas executem simultaneamente. O co-alocador é fundamental para execução em Grids de aplicações fortemente acopladas. Em aplicações fortemente acopladas, as tarefas precisam se comunicar para que a aplicação faça progresso. Portanto, todas as tarefas da aplicação têm que ser executadas simultanea-

mente. É importante ressaltar que uma boa implementação de co-alocação depende da implementação, por parte dos escalonadores de recurso, do serviço de reserva adiantadas (advance reservation). Reservas adiantadas permitem a escalonadores de aplicação obter garantias de escalonadores de recurso que determinados recursos (e.g. processadores) estarão disponíveis para aplicação em um intervalo de tempo preestabelecido [36].

A Figura 14 apresenta um exemplo da submissão de uma aplicação em um Grid Globus. Veja que um usuário envia uma solicitação de executar “uma simulação interativa envolvendo 100.000 entidades” para um escalonador de aplicação especializado em simulação interativa distribuída. Tal escalonador converte a solicitação original em outra mais especifica, que descreve a necessidade do usuário em termos de ciclos, memória e latência de comunicação. Esta nova solicitação é então enviada a um escalonador de aplicação especializado em MPPs. Este escalonador consulta o MDS para descobrir quais MPPs (dentro aqueles aos quais o usuário tem acesso) são os melhores para utilizar no momento. Além disso, o escalonador especializado em MPPs faz a partição do trabalho entre os MPPs escolhidos e envia a solicitação mais refinada para o co-alocador. O co-alocador garante que as tarefas submetidas aos distintos MPPs comecem a executar simultaneamente. Note também que outros escalonadores de aplicações podem participar do sistema. A Figura 14, por exemplo, exemplifica ainda escalonadores para varredura de parâmetros e para ambientes de colaboração virtual.

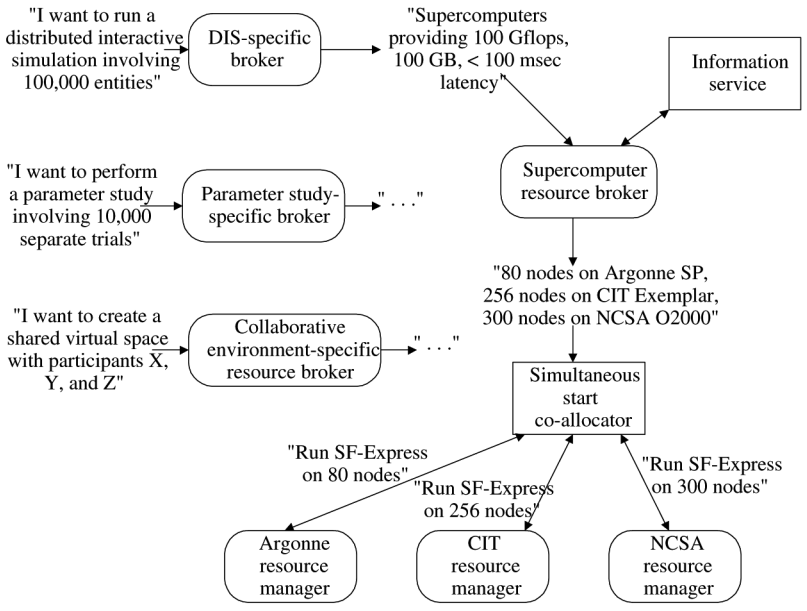

Figura 14 – Exemplo de submissão de aplicações a um Grid Globus [16]

Comunicação

O problema de comunicação no Grid pode ser visto como uma instância do eterno conflito entre generalidade e performance. Caso utilizemos um mecanismo de comunicação genérico (e.g. TCP) que viabilize a comunicação fim-a-fim entre quaisquer duas tarefas no Grid, perdemos performance em casos especiais (e.g. se ambas tarefas rodam em uma máquina de memória compartilhada, elas poderiam se comunicar muito mais rapidamente pela memória). Por outro lado, gostaríamos de usar um mecanismo genérico para não ter que programar para cada uma das várias tecnologias de comunicação existentes.

Globus ataca este problema com o Nexus. Nexus fornece uma interface de baixo nível, mas uma implementação adaptável que escolhe, dentre as tecnologias de comunicação disponíveis, a que vai oferecer melhor performance. Por exemplo, se ambas tarefas estão em uma máquina de memória compartilhada, Nexus utilizará a memória para efetuar a comunicação. Caso as tarefas estejam em um MPP, Nexus utilizará o switch de alta velocidade para comunicação. Caso as tarefas estejam em máquinas geograficamente distantes, Nexus utilizará TCP/IP.

Nexus fornece uma interface de relativo baixo nível: invocação remota de procedimento, mas sem retorno de resultado. Portanto, programar diretamente em Nexus não é das tarefas mais agradáveis. Entretanto, a idéia da equipe Globus é que Nexus seja usado por desenvolvedores de ferramentas e mecanismos de comunicação, não diretamente pelo desenvolvedor de aplicações. MPI-G é o exemplo perfeito desta abordagem. MPI-G implementa o popular padrão MPI (Message Passing Interface) sobre Nexus. Assim, o desenvolvedor de aplicações escrever em MPI e link-editar sua aplicação com MPI-G para automaticamente ter acesso a melhor tecnologia de comunicação disponível (selecionada pelo Nexus).

Transferência de Dados

A necessidade de acesso remoto e transferência de dados é uma constante na Computação em Grid. Na verdade, várias das aplicações aptas a executar no Grid necessitam de paralelismo exatamente porque processam enormes quantidades de dados. Ciente deste fato, Globus logo disponibilizou GASS (Global Access to Secundary Storage), um serviço para acesso remoto a arquivos sob a tutela de um servidor GASS. O cliente GASS é uma biblioteca C que é link-editada à aplicação usuária do serviço. Com o intuito de fornecer boa performance, o serviço GASS implementa as otimizações típicas de acesso remoto como caching e pre-fetching.

Apesar de ser um bom serviço, o GASS encontrou problemas de implantação. A dificuldade encontrada foi de interoperabilidade. A maioria das fontes de dados onde se instalaria um servidor GASS já executa algum serviço de transferência e/ou acesso remoto a arquivos. Os administradores de sistema se questionavam então porque não se poderia usar os serviços existentes.

Essa realidade motivou a introdução do GridFTP [2] por parte da equipe Globus. GridFTP estende o popular protocolo FTP para torná-lo mais adequado para as necessidades da Computação em Grid. Mais precisamente, GridFTP introduz suporte a:

- Autenticação GSI e Kerberos
- Transferência em paralelo (várias conexões TCP entre fonte e destino)
- Transferência striped (conexões TCP entre várias fontes e um destino, ou vice-versa)
- Controle manual dos buffers TCP (usado para afinamento de performance)
- Instrumentação embutida

Uma vez que GridFTP é uma extensão do FTP, o problema de interoperabilidade fica resolvido, pois FTP é amplamente suportado pelos servidores de dados. Obviamente, se as extensões GridFTP não estiverem implementadas em um dado servidor, os benefícios adicionais do protocolo não estarão disponível. Mas o cliente GridFTP ainda será capaz de obter os dados desejados.

Avaliação do Globus

Um aspecto importante para grande aceitação do Globus é que os serviços oferecidos são razoavelmente independentes, possibilitando que se utilize apenas parte dos serviços Globus em uma dada solução. Essa possibilidade do uso parcial de Globus ajuda sobremaneira na adaptação de aplicações paralelas existentes para o Grid. Pode-se começar usando serviços mais básicos e ir, aos poucos, incorporando funcionalidades mais avançadas. O design oposto à abordagem “conjunto de serviços independentes” do Globus é exemplificado pelo Legion [25]. Legion fornece um modelo orientado a objetos poderoso e flexível. Entretanto, o usuário tem abraçar a solução Legion integralmente, sem estágios intermediários. Esta diferença de abordagem talvez tenha contribuído para prevalência do Globus como padrão *de facto* como infraestrutura para Computação em Grid.

É interessante notar que a decisão de estruturar Globus como um conjunto de serviços independentes deixa claro que Globus *não* é uma solução pronta e completa (plug-and-play) para construção de Grids. Globus certamente fornece serviços úteis para Com-

putação em Grids. Mas, desenvolvedores, administradores e usuários precisam despender certo esforço para finalizar seu Grid. Por exemplo, administradores precisam decidir como quais usuários terão acesso a quais recursos que compõem o Grid e em quais condições este acesso se dará (veja seção 3.3). Em outro exemplo, freqüentemente é necessário desenvolver escalonadores de aplicação (veja seção 3.1) que tenham conhecimento sobre as aplicações que serão executadas e eventualmente também sobre a estrutura do Grid a ser usado.

Computação em Grid é simplesmente muito complexa para possibilitar soluções plug-and-play. Portanto, o fato do Globus não ser uma solução pronta e completa não é nenhum demérito. Entretanto, algumas pessoas têm a idéia de que Globus é *a solução*, completa e perfeita. Esta falsa concepção, sim, é um problema pois gera falsas expectativas e obscurece discussões técnicas com alegações de marketing.

Estado Atual

A versão atual do Globus (2.0) utiliza vários protocolos diferentes (LDAP no MDS, FTP e GridFTP para transferência de arquivos, etc) para implementação dos diversos serviços oferecidos. Na próxima versão (3.0), todos os serviços Globus serão baseados em Web Services, em um esforço para criar uma série de padrões para computação distribuída denominados Open Grid Services Architecture (OGSA) [19]. Esta evolução na direção de Web Services e OGSA mostra o interesse de seus desenvolvedores de utilizar Globus em contextos mais amplos que Processamento de Alto Desempenho (em particular, em processamento comercial pesado).

4.2 Condor

Condor é uma sistema que objetiva fornecer grande quantidade de poder computacional a médio e longo prazo (dias a semanas) utilizando recursos ociosos na rede [28]. Os autores do sistema salientam insistentemente que Condor objetiva alta vazão (high throughput) e não alto desempenho (high performance) [3] [14] [22] [28]. Entenda-se disto que Condor visa fornecer desempenho sustentável a médio e longo prazos, mesmo que o desempenho instantâneo do sistema possa variar consideravelmente.

O usuário submete ao Condor tarefas independentes para execução. Isso significa que uma aplicação Condor é uma aplicação Bag of Tasks. Enquanto este fato limita as aplicações que podem ser executadas em Condor, deve-se salientar que há um grande número de aplicações importantes que são Bag of Tasks. Além disso, aplicações Bag of Tasks, devido ao seu fraco acoplamento, são bastante adequadas para execução em Grids Computacionais.

Condor foi inicialmente concebido para funcionar em NOWs. Uma NOW que executa Condor denomina-se *Condor Pool*. O elemento arquitetural mais importante de um Condor Pool é o *Matchmaker*. O Matchmaker aloca tarefas a máquinas pertencentes ao Pool. Tal alocação é baseada nas necessidades de cada tarefa e nas restrições de uso de cada máquina. As necessidades de uma tarefa são especificadas pelo usuário quando de sua submissão. Por exemplo, uma tarefa pode precisar de uma máquina Sun Sparc, rodando Solaris, com pelo menos 256MB de memória. Já as restrições de uso de uma dada máquina, estas são especificadas por seu dono quando da inclusão da máquina no Pool. Por exemplo, o dono pode preferir que sua máquina execute as aplicações de João, seguido das aplicações do grupo de sistemas operacionais, e que nunca execute as aplicações de Pedro. Ou seja, as restrições permitem ao dono determinar como sua máquina será usada no Condor Pool. Tipicamente, o que o dono estabelece inclui que sua máquina só é usada quando estiver ociosa e que, quando ele voltar a utilizar a máquina, qualquer aplicação Condor em execução seja suspensa imediatamente.

Um aspecto interessante do Condor é que ambos usuários e donos de máquinas são representados no sistema por agentes de software. O Agente do Usuário (Customer Agent) envia as necessidades da tarefa para o Matchmaker. Similarmente, o Agente do Dono (Resource Owner Agent) envia as restrições de uso do recurso ao Matchmaker. Ao efetuar o casamento entre tarefa e máquina, o Matchmaker notifica ambos Agentes. A partir daí, o Agente do Usuário e o Agente do Dono interagem diretamente para realizar a execução da tarefa. A Figura 15 ilustra este protocolo.

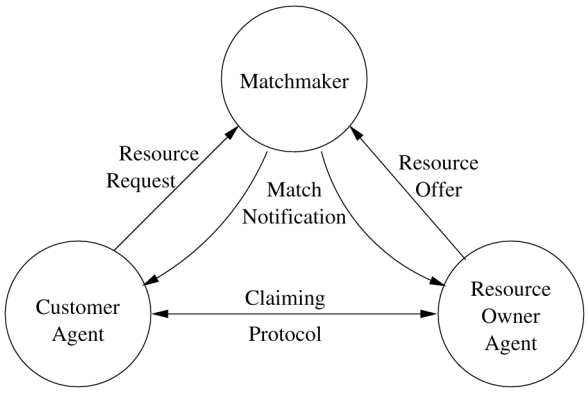

Figura 15 – Condor matchmaking [3]

Outro aspecto atraente do Condor é seu mecanismo de checkpointing. Uma vez que o dono normalmente especifica que sua máquina seja “desocupada” pela tarefa Con-

dor assim que ele retornar a usá-la, garantir progresso das tarefas torna-se um problema não-trivial. Condor aborda esta questão fazendo o checkpoint da tarefa (i.e. salvando transparentemente seu estado), o que permite que a tarefa seja re-executada em outra máquina a partir do ponto em que parou. O mecanismo de checkpoint do Condor é implementado através da substituição da biblioteca do sistema por uma biblioteca Condor. Note, portanto, que o mecanismo de checkpointing é implementado em conjunto com o mecanismo de redirecionamento de acesso a arquivos (descrito na seção 3.4). Na verdade, acesso aos arquivos na máquina base e migração de tarefas são complementares. De que adiantaria migrar tarefas se elas estivessem acessando o sistema de arquivos local?

Condor foi inicialmente concebido para execução em NOWs [28], mas posteriormente estendido para execução em Grids [14] [22]. O que tornou possível à extensão relativamente simples do Condor de NOWs a Grids, foi o fato da aplicação suportada (Bag of Tasks) ser bastante adequada a Grids Computacionais. É interessante notar que Condor dispõe de duas formas de funcionamento em Grids: *Flock of Condors* [14] e *Condor-G* [22].

Flock of Condors é um trabalho que antecede Condor-G. Um Flock of Condors é um Grid formado por vários Condor Pools [14]. A construção do Flock é bastante elegante do ponto de vista de sistemas distribuídos pois não acrescenta nenhuma centralização a arquitetura Condor original. A base para criação de um Flock é um acordo de cooperação (de troca de recursos) entre *dois* Condors Pools. Portanto, por maior que seja o Flock, suas ligações são sempre dois-a-dois, sem envolver nenhuma entidade centralizadora. Mais que isso, o Flock of Condors não chega a alterar o software Condor original. Todo a funcionalidade do Flock of Condors é implementada por uma máquina especial, chamada *Gateway*. Ambos os Pools que firmam um acordo de cooperação instalam cada qual um Gateway. Os dois Gateways mantém comunicação constante para troca de tarefas entre os Pools. Para o Pool local, o Gateway é uma máquina qualquer. Entretanto, ao invés de oferecer seus próprios recursos, o Gateway simplesmente representa os recursos do Pool remoto, republicado as restrições estabelecidas pelos donos das máquinas remotas. Quando uma tarefa é recebida pelo Gateway, este a repassa para o Gateway remoto que então a encaminha para uma máquina para execução.

Talvez por ser mais recente, o Condor-G adota uma visão mais heterogênea de Grid. Além de Condor Pools, Condor-G também utiliza recursos via Globus (veja seção 4.1). Devido à necessidade de suportar mais heterogeneidade, Condor-G usa uma arquitetura mais centralizada que o Flock of Condors. O Condor-G Scheduler controla a exe-

cução da aplicação, submetendo tarefas tanto a Condor Pools quanto a recursos acessíveis via Globus (como MPPs). No caso de recursos Globus, Condor-G utiliza os serviços GRAM, GASS, MDS e GSI (veja seção 4.1) para viabilizar a execução das tarefas.

4.3 MyGrid

Embora bastante pesquisa tenha sido realizada recentemente em Computação em Grid, poucos usuários atualmente executam suas aplicações paralelas sobre Grids. Concebemos o projeto MyGrid com o intuito de alterar esta situação. Para tanto, atacamos apenas aplicações Bag of Tasks, ou seja, aquelas aplicações cujas tarefas são independentes, podendo ser executadas em qualquer ordem. Aplicações Bag of Tasks são um alvo interessante porque (i) se adequam melhor a ampla distribuição, heterogeneidade e dinamicidade do Grid, e (ii) resolvem vários problemas importantes, tais como mineração de dados, processamento genômico, pesquisa massiva (como quebra de chaves criptográficas), varredura de parâmetros, simulações Monte Carlo, computação de fractais (como Mandelbrot), e manipulação de imagem (como tomografia).

Além de nos restringirmos a aplicações Bag of Tasks, evitaremos também a questão de como formar o Grid (veja discussão na seção 3.3). Para MyGrid, o Grid de um dado usuário é composto por *todas as máquinas que o usuário pode acessar*. O Grid de um dado usuário pode conter as máquinas de seu laboratório, de outros laboratórios com que o usuário desenvolve atividades conjuntas, de algum provedor contratado para fornecer ciclos, e até de algum amigo que forneceu um login para sua máquina.

Note que a decisão de definir Grid em função do que o usuário pode acessar se afina bem com o fato de suportarmos aplicações Bag of Tasks. Devido à fraca acoplamento entre as tarefas de uma aplicação Bag of Tasks, o usuário pode em principio se beneficiar de quaisquer processadores que ele tenha acesso. Não queremos, portanto, uma solução para aplicações Bag of Tasks que impeça o usuário de usar algum processador por ele não pertencer “ao Grid”.

Estabelecido o escopo do MyGrid, nosso objetivo é construir um sistema *simples*, *completo* e *seguro*. Por *simples* queremos dizer que o esforço para utilização do MyGrid deve ser mínimo. Em particular, queremos chegar o mais próximo possível de uma solução pronta (plug-and-play). Por *completo* denotamos a necessidade de cobrir todo o ciclo de uso de um sistema computacional, do desenvolvimento à execução, passando por instalação e atualização e incluindo também a manipulação de arquivos. Por *seguro* expressamos a necessidade de não introduzir vulnerabilidades ao ambiente computacional do usuário. Ou seja, não queremos que falhas de segurança em qualquer uma das má-

quinas que o usuário possa utilizar sejam propagadas para sua máquina base (i.e. o computador usado pelo usuário).

Arquitetura

MyGrid diferencia entre *máquina base* e *máquina do grid*. Em um MyGrid, a *máquina base* é aquela que controla a execução da aplicação. Ela tipicamente contém os dados de entrada e coleta os resultados da computação. A máquina base é normalmente usada pelo usuário diretamente no seu dia-a-dia, muitas vezes sendo o próprio computador desktop do usuário. Esperamos, portanto, que o usuário tenha excelente acesso à máquina base e que tenha customizado um ambiente de trabalho confortável nela.

Todas as máquinas usadas via MyGrid para executar tarefas são chamadas de *máquinas de grid*. Ao contrário da máquina base, não assumimos que o usuário customizou cada máquina do grid para criar-lhe um ambiente de trabalho familiar. Além disso, todas as máquinas do grid tipicamente não compartilham um mesmo sistema de arquivo ou têm os mesmos softwares instalados. A imagem do sistema pode variar de uma máquina do grid para outra. Portanto, para mantermos a simplicidade de uso do sistema, precisamos evitar que o usuário tenha que lidar diretamente com as máquinas do grid. Por exemplo, queremos evitar que o usuário tenha que instalar software em cada máquina de seu Grid. A idéia é que máquinas do grid sejam manipuladas através das abstrações criadas por MyGrid (descritas a seguir).

Um aspecto importante de MyGrid é dar ao usuário a possibilidade de usar “quaisquer recursos que ele tenha acesso”. Este não é um objetivo trivial porque ele implica que temos que assumir muito pouco a respeito de uma máquina do grid, de forma a não impedir algum usuário de usar uma máquina que não suporta nossas hipóteses. Em particular, não podemos assumir que tal recurso tenha software MyGrid instalado. MyGrid define *Grid Machine Abstraction* como sendo o conjunto mínimo de serviços que precisam estar disponíveis para que uma dada máquina possa ser adicionada ao grid do usuário. Tais serviços são:

Serviço
Execução remota
Transferência de arquivos da máquina do grid para a máquina base
Transferência de arquivos da máquina base para a máquina do grid

Tabela 3 – Serviços que definem a Grid Machine Abstraction

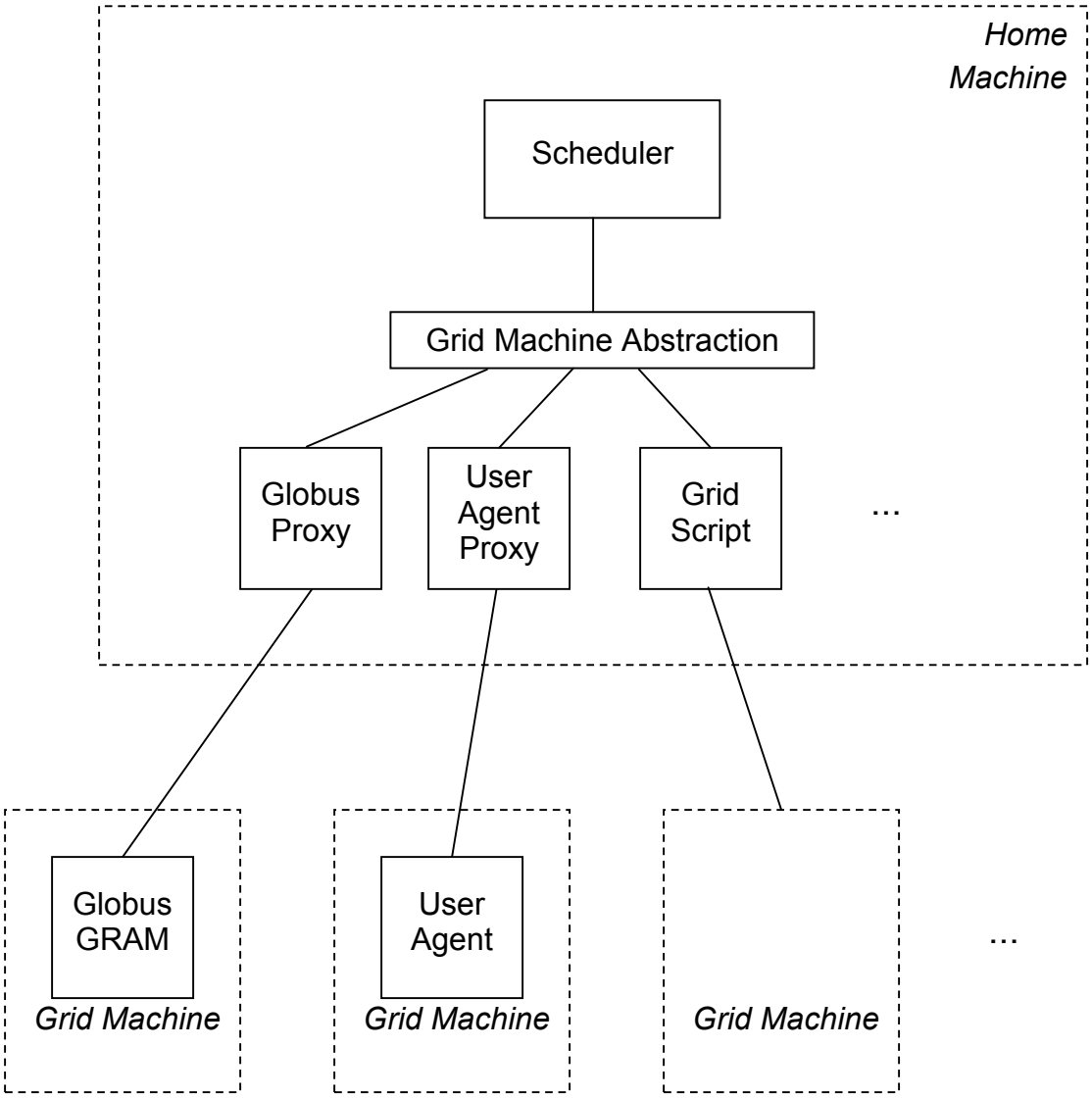

Figura 16 – Arquitetura do MyGrid

Como ilustrado pela Figura 16, há várias formas de implementar a Grid Machine Abstraction. Uma forma é fornecer ao sistema scripts que implementam os três serviços listados na Tabela 3. Neste caso, MyGrid utiliza o módulo *Grid Script* para acessar a máquina em questão. Note que Grid Script possibilita que, qualquer que seja a máquina que o usuário tem acesso, ele possa informar como este acesso se dá através da escrita de três scripts. Alternativamente, há casos em que a forma de acessar uma determinada máquina do grid já é do conhecimento do MyGrid. Por exemplo, suponha que a máquina em questão pode ser acessa via serviços Globus (GSI, GRAM e GridFTP). Neste caso, o usuário não precisa fornecer os scripts, indicando apenas que o acesso à máquina já é conhecido de MyGrid. Finalmente, MyGrid também provê um mecanismo de acesso a máquinas do grid, chamado de *User Agent*. O User Agent provê serviços simples (aque-

les descritos na Tabela 3 como também algum suporte especializado para simplificar criação das abstrações MyGrid). É interessante notar que, pela terminologia de [19], *Grid Machine Abstraction* é uma *virtualização* para os serviços de acesso a uma máquina do Grid.

Outro componente fundamental a arquitetura MyGrid é o Scheduler. O Scheduler recebe do usuário a descrição das tarefas a executar, escolhe qual processador usar para cada tarefa, e finalmente submete e monitora a execução da tarefa. Por obter boa performance mesmo sem informações sobre o estado do Grid ou o tamanho de cada tarefa, o algoritmo usado pelo Scheduler é Work Queue with Replication (veja seção 3.1). Note que ter um algoritmo de escalonamento que funciona bem *sem depender de muita informação* é importante pois simplifica a definição da Grid Machine Abstraction. Caso o algoritmo de escalonamento do MyGrid necessitasse de informações sobre as máquinas do grid, Grid Machine Abstraction teria que ser mais rica e, portanto, mais difícil de virtualizar.

Usando MyGrid

Uma tarefa MyGrid é composta por três partes ou sub-tarefas: *inicial*, *grid* e *final*, que são executadas seqüencialmente, nesta ordem. Sub-tarefas são comandos externos invocados por MyGrid e podem portanto executar qualquer programa, escrito em qualquer linguagem. Tanto a sub-tarefa inicial quando a final são executadas na máquina base. A sub-tarefa inicial é tipicamente usada para transferir a entrada para máquina do grid. Já a sub-tarefa final foi concebida para que os resultados possam ser recolhidos para a máquina base. A sub-tarefa grid é executada em uma máquina do grid e realiza a computação propriamente dita da tarefa.

Quando da escrita das sub-tarefas inicial e final, o usuário lança mão das abstrações MyGrid para lidar com a máquina do grid sem necessitar de conhecer sua imagem do sistema. As abstrações MyGrid são *espelhamento*, *playpens* e *transferência de arquivos*. O serviço de espelhamento possibilita ao usuário reproduzir nas máquinas do grid arquivos que ele tem na sua máquina base. Espelhamento é útil para distribuição de arquivos que provavelmente serão reutilizados no grid (como binários). Assim sendo, qualquer arquivo espelhado é automaticamente incluído no `PATH` da máquina do grid. Espelhamento é implementado eficientemente através do uso de versões para evitar a transferência desnecessária de arquivos. Playpens, por outro lado, provêm espaço temporário em disco de maneira independente das convenções locais do sistema de arquivo de uma dada máquina do grid. Playpens são implementados pela criação de um diretório que pode conter a quantidade de dados especificada pelo usuário. Eles são criados automati-

camente e são o diretório em que a sub-tarefa grid é executada. Note que playpens foram concebidos para possibilitar o armazenamento de dados temporários e resultados de tarefas. Transferência de arquivo obviamente possibilita a troca de arquivos entre a máquina base e as máquinas do grid. Também no sentido de simplificar a escrita das sub-tarefas, as variáveis de ambiente $PROC, $PLAYPEN e $TASK são automaticamente definidas pelo MyGrid e contêm, respectivamente, o nome da máquina do grid escolhida para executar a tarefa, o nome do diretório playpen e o número único da tarefa.

Por exemplo, suponha que queremos rodar o executável tarefa, que tem como entrada o arquivo ENTRADA e produz o arquivo SAÍDA. A sub-tarefa inicial seria portanto:

```
mg-services mirror $PROC tarefa
mg-services put $PROC ENTRADA $PLAYPEN
```

A sub-tarefa grid seria algo como:

```
tarefa < ENTRADA > SAÍDA
```

E a sub-tarefa final recolheria a saída da tarefa para o diretório resultados.

```
mg-services get $PROC $PLAYPEN/SAÍDA resultados/SAÍDA
```

5 Conclusões e Perspectivas Futuras

Computação em Grid é uma área recente e extremamente dinâmica. Grande parte do interesse em Computação em Grid advém do potencial de atingir níveis de paralelismo inimagináveis em outras plataformas de execução de aplicações paralelas. Tais níveis de paralelismo podem se traduzir em tanto mais performance para aplicações existentes, quanto também a possibilidade de executar aplicações inteiramente novas, com gigantescos requisitos de computação e armazenamento. Além disso, a idéia de usar Grids como laboratórios virtuais, que viabilizam colaborações cientificas a distância é das mais animadoras, com potencial impacto no nosso modus operandi como cientistas.

Mas, além de tão grande avanço técnico para a área de Processamento de Alto Desempenho (PAD), Computação em Grid é também muito interessante no que diz respeito à convergência entre tecnologias de PAD e computação empresarial. Por exemplo, Globus 3.0 [19] está sendo atualmente desenvolvido com apoio da IBM e se baseia fundamentalmente em Web Services, uma tecnologia concebida para uso empresarial. Isto mostra o interesse dos desenvolvedores Globus de utilizar as soluções desenvolvidas para Grids “científicos” em contextos mais amplos que PAD, como também o interesse da industria (no caso, IBM) de utilizar tecnologia Grid em computação empresarial (enterprise computing).

Como não poderia ser diferente, além das questões de pesquisa a serem resolvidas para a tornar realidade a visão de Computação em Grid, há também muito trabalho a ser feito na área de padronização. Em particular, não faz sentido ter soluções estanques para grande parte da infraestrutura necessária para Computação em Gris, como serviços de informação e certificação de chave-pública. Felizmente, foi criado o Grid Forum [24] com o objetivo de viabilizar a discussão destes padrões, como também fomentar a área de Computação em Grid como um todo.

Quanto a aspectos técnicos propriamente ditos, como Computação em Grid ainda está em sua infância, há diversas questões em aberto na área. Em particular, acreditamos que progresso é urgentemente necessário (i) na criação de modelos de programação que melhor exponham a natureza do Grid ao programador, e (ii) em melhores formas de lidar com grandes massas de dados no Grid. Grids são mais complexos e dinâmicos que outras plataformas para execução de aplicações paralelas. Se quisermos garantir performance é necessário conceber abstrações úteis, que permitam o programador codificar estratégias de adaptação à complexidade do Grid.

Em relação a grandes volumes de dados no Grid (um assunto que vem sendo chamado de Data Grid), julgamos esta área crucial porque ter que processar muitos dados parece ser uma das mais fortes motivações para utilização de Grids. A quantidade de dados gerados pela humanidade vem crescendo de forma exponencial e tal efeito tende a se manter, pelo menos a médio prazo. Usar paralelismo para processar vastas quantidades de dados é bastante natural. Além disso, as soluções geradas são normalmente fracamente acopladas, sendo portanto adequadas para execução em Grids. É verdade que temos visto considerável atividade em Data Grid nos últimos 2 anos [1] [6] [10] [13] [27]. Entretanto, ainda há muito o que se fazer para que o grande potencial do uso de Grids para aplicações massivas em dados seja efetivamente “colocado em produção”.

Referências

[1] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke. *Data Management and Transfer in High Performance Computational Grid Environments*. Parallel Computing Journal, Vol. 28 (5), May 2002, pp. 749-771.
http://www.globus.org/research/papers.html

[2] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, S. Tuecke. *GridFTP Protocol Specification*. GGF GridFTP Working Group Document, September 2002.
http://www.globus.org/research/papers.html

[3] Jim Basney and Miron Livny. *Deploying a High Throughput Computing Cluster*. High Performance Cluster Computing, Rajkumar Buyya, Editor, Vol. 1, Chapter 5, Prentice Hall PTR, May 1999.
http://www.cs.wisc.edu/condor/publications.html

[4] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. *Application-Level Scheduling on Distributed Heterogeneous Networks*. Supercomputing'96, 1996.
http://apples.ucsd.edu/hetpubs.html

[5] R. Buyya, D. Abramson, J. Giddy. *An Economy Driven Resource Management Architecture for Global Computational Power Grids*. The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), Las Vegas, USA, June 26-29, 2000.
http://www.cs.mu.oz.au/~raj/ecogrid/

[6] H. Casanova, A. Legrand, D. Zagorodnov e F. Berman. *Heuristics for Scheduling Parameter Sweep applications in Grid environments*. In Proceedings of the 9th Heterogeneous Computing workshop (HCW'2000), pp349-363.
http://apples.ucsd.edu/hetpubs.html

[7] W. Cirne e K. Marzullo.*The Computacional Co-op: Gathering Clusters into a Metacomputer*. Proceeding of the IPPS/SPDP'99 Symposium. April 1999.
http://walfredo.dsc.ufcg.edu.br/resume.html#publications

[8] W. Cirne e K. Marzullo. *Open Grid:* A *User-Centric Approach for Grid Computing*. Proceedings of the 13th Symposium on Computer Architecture and High Performance Computing, September 2001.
http://walfredo.dsc.ufcg.edu.br/resume.html#publications

[9] W. Cirne, D. Paranhos, E. Santos-Neto, L. Costa, F. Brasileiro, C. Osthoff e F. Silva. *Running Bag of Tasks Applications on Computational Grids*. Submetido para publicação.
http://walfredo.dsc.ufcg.edu.br/resume.html#publications

[10] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. *The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets*. Journal of Network and Computer Applications, 23:187-200, 2001
http://www.globus.org/research/papers.html

[11] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. A *Resource Management Architecture for Metacomputing Systems*. Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, po. 62-82, 1998.
http://www.globus.org/research/papers.html

[12] distributed.net Web Page. http://www.distributed.net/index.html.en

[13] Wael Elwasif, James S. Plank and Rich Wolski. *Data Staging Effects in Wide Area Task Farming Applications*. IEEE International Symposium on Cluster Computing and the Grid, Brisbane, Australia, May, 2001, pp. 122-129.
http://www.cs.utk.edu/~plank/plank/papers/CC-GRID-01.html

[14] D. H. Epema, Miron Livny, R. van Dantzig, X. Evers, and Jim Pruyne. *A Worldwide Flock of Condors: Load Sharing among Workstation Clusters*. Journal on Future Generations of Computer Systems, Volume 12, 1996
http://www.cs.wisc.edu/condor/publications.html

[15] Entropia Web Page. http://www.entropia.com/

[16] I. Foster and C. Kesselman. *The Globus Project:* A *Status Report.* Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop*,* pg. 4-18, 1998.
http://www.globus.org/research/papers.html

[17] I. Foster and C. Kesselman (editors). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers. 1999.

[18] I. Foster, C. Kesselman, and S. Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. International Journal of Supercomp. Applications, 15(3), 2001.
http://www.globus.org/research/papers.html

[19] I. Foster, C. Kesselman, J. Nick, S. Tuecke. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. June 22, 2002.
http://www.globus.org/research/papers.html

[20] I. Foster. *What is the Grid? A Three Point Checklist*. GRID today, vol. 1, no. 6, 22 de julho de 2002.
http://www.gridtoday.com/02/0722/100136.html

[21] Paul Francis, Sugih Jamin, Vern Paxson, Lixia Zhang, Daniel Gryniewicz, and Yixin Jim. *An Architecture for a Global Internet Host Distance Estimation Service*. Proceedings of IEEE INFOCOM, 1999.

[22] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke. *Condor-G: A Computation Management Agent for Multi-Institutional Grids*. Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10), San Francisco, California, August 7-9, 2001.
http://www.cs.wisc.edu/condor/publications.html

[23] Globus Web Page. http://www.globus.org/

[24] Grid Forum Web Page. http://www.gridforum.org/

[25] A. Grimshaw and W. Wulf. *Legion: The next logical step toward the world-wide virtual computer*. Communications of the ACM, 40(1):39-45, January 1997.
http://citeseer.nj.nec.com/article/grimshaw96legion.html

[26] T. Hogg and B. Huberman. *Controlling Chaos in Distributed Systems*. IEEE Transactions on Systems, Man, and Cybernetics, vol. 21, no. 6, Nov/Dec 1991.

[27] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. *OceanStore: An Architecture for Global-Scale Persistent Storage*. Proceedings of ACM ASPLOS, Boston, MA, November 2000.
http://oceanstore.cs.berkeley.edu/publications/papers/pdf/asplos00.pdf

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

[29] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. *A Resource Query Interface for Network-Aware Applications*. Seventh IEEE Symposium on High-Performance Distributed Computing, July 1998.
http://www.cs.cmu.edu/~cmcl/remulac/papers.html

[30] M. Mitzenmacher. *How Useful is Old Information?* Proc. of the 16th Annual ACM Sym-posium on Principles of Distributed Computing (PODC 97), 1997. Also in IEEE Transaction in Parallel and Distributed Systems, 11(1), pg. 6-20, Jan. 2000.

[31] MyGrid Web Page. http://www.dsc.ufcg.edu.br/mygrid

[32] D. Paranhos, W. Cirne e F. Brasileiro. *Trading Information for Cycles: Using Replication to Sched-ule Bag of Tasks Applications on Computational Grids*. Submetido para publicação.
http://walfredo.dsc.ufcg.edu.br/resume.html#publications

[33] C. De Rose e P. Navaux. *Fundamentos de Processamento de Alto Desempenho*. ERAD 2002 – 2ª Escola Regional de Alto Desempenho, São Leopoldo, RS, Brasil, 15 a 19 de janeiro de 2002.

[34] SETI@home Web Page. http://www.seti.org/science/setiathome.html

[35] Gary Shao, Rich Wolski, and Fran Berman. *Predicting the Cost of Redistribution in Scheduling*. In Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing, 1997.
http://www.cs.ucsd.edu/groups/hpcl/apples/hetpubs.html

[36] W. Smith, I. Foster, V. Taylor. *Scheduling with Advanced Reservations*. Proceedings of the IPDPS Conference, May 2000.
http://www.globus.org/research/papers.html

[37] TeraGrid Web Page. http://www.teragrid.org/

[38] A. Vahdat, P. Eastham, C. Yoshikawa, E. Belani, T. Anderson, D. Culler, and M. Dahlin. *WebOS: Operating system services for wide area applications*. Proceedings of the Seventh Symposium on High Performance Distributed Computing, 1998.
http://citeseer.nj.nec.com/vahdat97webos.html

[39] C. Waldspurger e William Weihl. Stride Scheduling: Deterministic Proportional-Share Resource Mangement. Technical Memorandum MIT/LCS/TM-528, MIT Laboratory for Computer Science, June 1995.
http://www.research.digital.com/SRC/personal/caw/papers.html

[40] Jon Weissman and Andrew Grimshaw. *A Framework for Partitioning Parallel Computations in Heterogeneous Environments*. Concurrency: Practice and Experience, Vol. 7, No. 5, August 1995.
http://ringer.cs.utsa.edu/faculty/weissman.html/pub.html

[41] Jon Weissman. *Gallop: The Benefits of Wide-Area Computing for Parallel Processing*. Journal of Parallel and Distributed Computing, Vol. 54(2), November 1998.
http://ringer.cs.utsa.edu/faculty/weissman.html/pub.html

[42] R. Wolski, N. Spring, and J. Hayes. *The Network Weather Service:* A *Distributed Resource Performance Forecasting Service for Metacomputing*. Journal of Future Generation Computing Systems, 1998.

http://www.cs.utk.edu/~rich/publications/index.html

[43] Huican Zhu et al. *Adaptive Load Sharing for Clustered Digital Library Servers*. Seventh IEEE International Symposium on High Performance Distributed Computing, Chicago, Illinois, July, 1998.
http://www.alexandria.ucsb.edu/~zheng/publications.html