Uma Arquitetura de Redes Virtuais para um Grid Aberto e Seguro

Leonardo Assis, Matheus Gaudencio, Elizeu Santos-Neto, Francisco Brasileiro

[1]Universidade Federal de Campina Grande
Departamento de Sistemas e Computação
Laboratório de Sistemas Distribuídos
Av. Aprígio Veloso, 882, Bloco CO
58.109-970, Campina Grande, PB, Brasil

{leonardo,matheusgr,elizeu}@lsd.ufcg.edu.br

fubica@dsc.ufcg.edu.br

Abstract. *In the past, any computational grid was related with loosely coupled applications, which limited the potential systems that could be used with a grid. With advances on the related technology and with the grid potential to aggregate resources on demand and at low-cost, we expect that more tightly coupled applications will be executed in a computational grid. But, mainly due to security restrictions running thighted coupled applications across different administrative domains still presents some challenges. This work presents a new architecture to on demand deployment of overlay virtual network in order to enable the execution of coupled applications on a a set of resources geographically dispersed. Also, it keeps the resource protected from malicious applications. The OurGrid middleware was used to provide this system that is capable to execute tightly coupled applications with resources from different administrative domains.*

Resumo. *No início das pesquisas em grids computacionais, um grid computacional era amplamente associado a aplicações desacopladas, onde as partes constituintes da aplicação não trocavam informação entre si. Atualmente, devido aos avanços tecnológicos e com a possibilidade de se utilizar recursos sob-demanda e com baixo custo, espera-se cada vez mais que grids computacionais sejam usados para execução de aplicações com algum nível de acoplamento. Entretanto, devido a restrições de segurança, executar aplicações acopladas através de vários domínios administrativos diferentes ainda é um desafio. Este trabalho apresenta a arquitetura de um mecanismo para implantação de redes virtuais sob demanda no intuito de permitir a execução de aplicações fortemente acopladas em um grid computacional sem comprometer a segurança dos recursos. Essa arquitetura foi implementada no OurGrid (um middleware de grid aberto) para permitir a execução de aplicações acopladas utilizando recursos localizados em diferentes sites.*

1. Introdução

A pesquisa em grids computacionais tem recebido cada vez mais destaque. No início, a tecnologia estava amplamente associada a aplicações totalmente desacopladas ou apenas fracamente acopladas. Isto é, aplicações em que a comunicação entre as partes que as

constituem não trocam qualquer informação, ou fazem isso com uma taxa muito baixa. Qualquer uso que exigisse um maior acoplamento era reservado para recursos com boa conectividade (e.g. em um mesmo site). Atualmente com os grids computacionais sendo capazes de obter mais recursos a um menor custo e sob demanda, espera-se que cada vez mais estes sejam usados para execução de aplicações fortemente acopladas.

Entretanto, devido principalmente às restrições de segurança, executar aplicações fortemente acopladas através de múltiplos domínios administrativos não é uma tarefa trivial. Por exemplo, *firewalls* e NATs (*Network Address Translators*) criam uma barreira administrativa que complica a implantação de aplicações paralelas através de múltiplos sites, cada um com suas respectivas políticas de segurança.

Considerando a potencialidade do uso de grids computacionais para aplicações de diferentes graus de acoplamento, nós argumentamos que é relevante ter mecanismos que tornam possível a execução dessas aplicações em uma coleção de recursos situados em diferentes sites. Para que isto aconteça, é preciso que tais aplicações funcionem mesmo com a assimetria da rede imposta por *firewalls* e NATs sem, no entanto, diminuir a segurança exigida pelos diferentes domínios administrativos.

Este trabalho foi desenvolvido no contexto do OurGrid: um middleware de grid aberto que suporta originalmente apenas aplicações desacopladas do tipo "saco de tarefas" (Bag-of-Tasks, ou BoT) [Cirne et al. 2006]. O mecanismo básico de segurança utilizado pelo OurGrid é baseado no conceito de *sandboxing*, que protege os recursos disponibilizados no grid de aplicações hostis através da criação de uma máquina virtual que isola a computação remota do hardware e do restante do software executando no recurso [San 2007]. Em particular, a arquitetura atual do OurGrid se utiliza do fato de que apenas aplicações desacopladas são executadas para impedir que a computação remota possa ter acesso à rede, limitando assim a possibilidade de um ataque ser desencadeado usando o recurso que hospeda a computação remota.

O objetivo principal deste trabalho é estender a arquitetura do OurGrid para permitir a execução de aplicações acopladas agregando recursos de diferentes domínios administrativos sem no entanto comprometer as premissas de segurança. Para tal, nós usamos a tecnologia de redes virtuais (em particular Xen VNET [Wray 2006, Kallahalla et al. 2004]) combinada com um mecanismo de implantação que configura uma topologia virtual sob demanda.

O restante deste artigo está organizado da seguinte forma. A Seção 2 descreve a arquitetura atual do OurGrid. Na Seção 3 é descrita a arquitetura estendida proposta do OurGrid que possibilita a criação sob-demanda e automática de redes virtuais capazes de executarem aplicações paralelas fortemente acopladas utilizando recursos de diferentes domínios administrativos. Os trabalhos relacionados com a nossa arquitetura são descritos na Seção 4, enquanto que a Seção 5 conclui o artigo com os nossos comentários finais e uma discussão sobre direções futuras dessa pesquisa.

2. Arquitetura do OurGrid para Aplicações Desacopladas

O OurGrid é composto de três componentes principais: OG Peer, OG Broker e SWAN Worker. Cada uma destas entidades tem um comportamento e papéis bem definidos na arquitetura do middleware, como descritos a seguir:

OG Peer Gerencia os recursos do site e obtêm recursos ociosos de outros sites para os clientes locais e também doa os recursos locais ociosos para outros OG Peers;

OG Broker Fornece uma interface para submissão de aplicações (ou *jobs*, na terminologia OurGrid), que são formadas por um conjunto de tarefas que podem ser executadas de forma paralela no grid; é também tarefa do OG Broker executar o escalonamento das tarefas que compõem as aplicações nos recursos do grid alocados para cada aplicação.

SWAN Worker Um mecanismo de *sandboxing* baseado no *hypervisor* Xen. A abstração fornecida pelo SWAN Worker consiste de um recurso seguro que irá executar tarefas em uma máquina virtual, protegendo o recurso contra uma possível aplicação maliciosa.

A Figura 1 mostra a arquitetura do OurGrid.

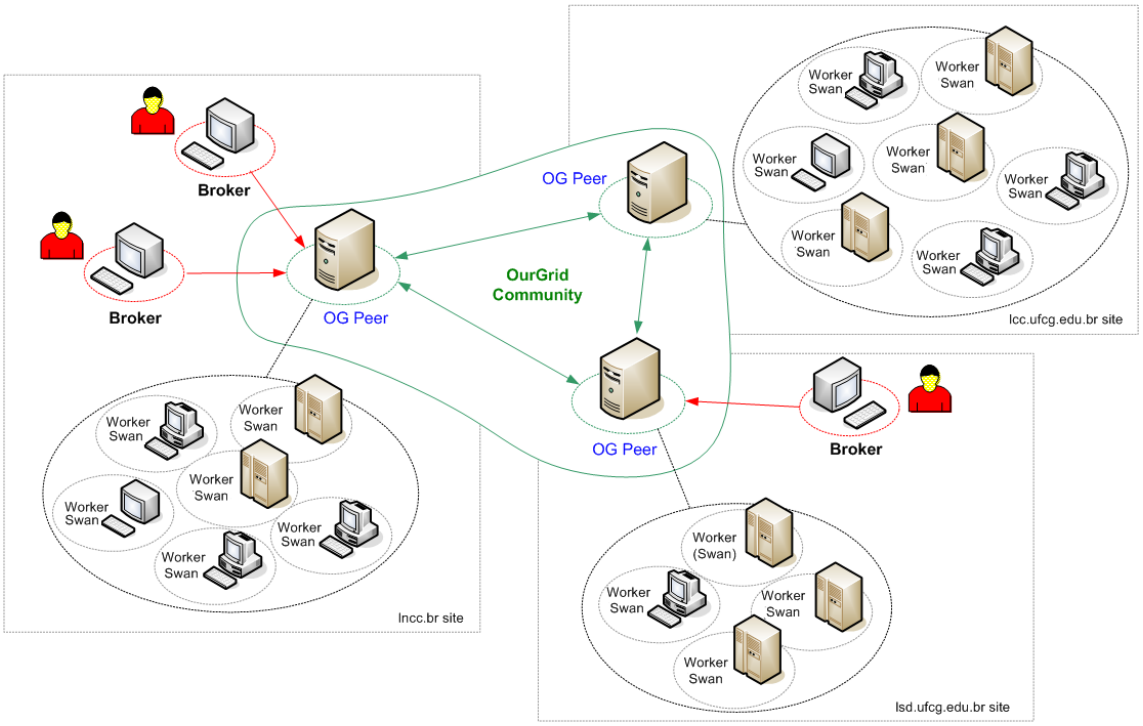

Figura 1. Componentes do OurGrid

2.1. Funcionamento do OurGrid

Considerando um cenário composto por *OG Peer A* com as máquinas *SWAN Worker A1*, *SWAN Worker A2*; *OG Peer B* com a máquina *SWAN Worker B1*; *OG Peer C* com as máquinas *SWAN Worker C1*, *SWAN Worker C2*; a cientista Eva, usando o seu *OG Broker Eva*; o cientista Bob, usando o seu *OG Broker Bob*, descreveremos o caso de uso onde a cientista Eva deseja executar um job.

Inicialmente, *OG Broker Eva* é conectado ao *OG Peer A*. A submissão de um job ocorre então da seguinte maneira:

1. Eva submete um job ao *OG Broker Eva*;
2. *OG Broker Eva* requisita máquinas ao *OG Peer A*;

3. *OG Peer A* fornece o máximo de máquinas locais disponíveis para *OG Broker Eva*;
4. *OG Peer A* pede recursos para *OG Peer B* e *OG Peer C*;
5. *OG Peer B* e *OG Peer C* fornecem as máquinas ociosas para *OG Peer A*;
6. *OG Peer A* envia as máquinas para o *OG Broker Eva*;
7. *OG Broker Eva* escalona as tarefas para as máquinas recebidas até que todas as tarefas sejam executadas, efetuando assim a execução do job.

É importante observar que se Bob requisitasse máquinas ao *OG Peer B* via *OG Broker Bob*, a máquina doada a Eva *SWAN Worker B1* seria transferida para Bob. Essa política de alocação garante que nunca é pior estar no grid, pois o usuário sempre tem prioridade sobre seus recursos locais. Além disso, o OurGrid implementa um mecanismo de incentivo à doação de recursos chamado de rede de favores [Santos et al. 2005, Andrade et al. 2004]. Quando *OG Peer C* doa uma máquina ociosa ao *OG Peer A*, o primeiro está fazendo um favor para o último, na expectativa que no futuro *OG Peer A* irá retribuir esse favor. No exemplo acima, *OG Peer B* também poderia solicitar máquinas ao *OG Peer C*; dependendo do saldo de favores que o *OG Peer B* e *OG Peer A* têm com o *OG Peer C*, este pode decidir transferir a máquina alocada ao *OG Peer A* para o *OG Peer B*. Dessa forma, os peers são incentivados a doarem seus recursos ociosos.

2.2. SWAN Worker

Para entender o SWAN Worker, é preciso entender como funciona a máquina virtual Xen. O Xen é um monitor de máquinas virtuais que opera através de paravirtualização, ou seja, um *hypervisor* [Barham et al. 2003]. Ao iniciar uma máquina com Xen, se está, na verdade, iniciando um microkernel Xen responsável por controlar as máquinas virtuais do sistema, e também kernels Linux modificados para funcionar com este hypervisor. O Xen suporta dois tipos especiais de kernel Linux: domain0 (um domínio confiável) e domainU (domínio não-confiável). Na Figura 2 é mostrada a arquitetura do SWAN Worker.

O que diferencia estes dois tipos de kernel é que o domínio confiável terá acesso ao hardware real da máquina bem como a chamadas de controle do hypervisor. O não-confiável por sua vez terá um sistema virtual e restrito, sem possibilidade de controlar o hypervisor.

Assim, considerando uma tarefa de Eva a ser executada no OurGrid. Dada uma tarefa que foi escalonada pelo *OG Broker Eva* para ser executada em *SWAN Worker C1*:

1. O *OG Broker Eva* envia os dados referentes a tarefa para *SWAN Worker C1*;
2. O *SWAN Worker C1* armazena os dados recebidos em partições especiais;
3. O *OG Broker Eva* ordena o começo da execução;
4. O *SWAN Worker C1* cria uma máquina virtual com acesso às partições que receberam os dados;
5. O *SWAN Worker C1* espera a máquina virtual finalizar sua execução;
6. O *SWAN Worker C1* recupera os dados gerados pela aplicação e os envia para *OG Broker Eva*.

Na solução atual do SWAN Worker, a máquina virtual criada tem acesso apenas a partições temporárias que são apagadas após cada execução, além de acesso restrito à rede onde os *SWAN Workers* de um mesmo site compartilham uma área chamada *storage* montada através do NFS.

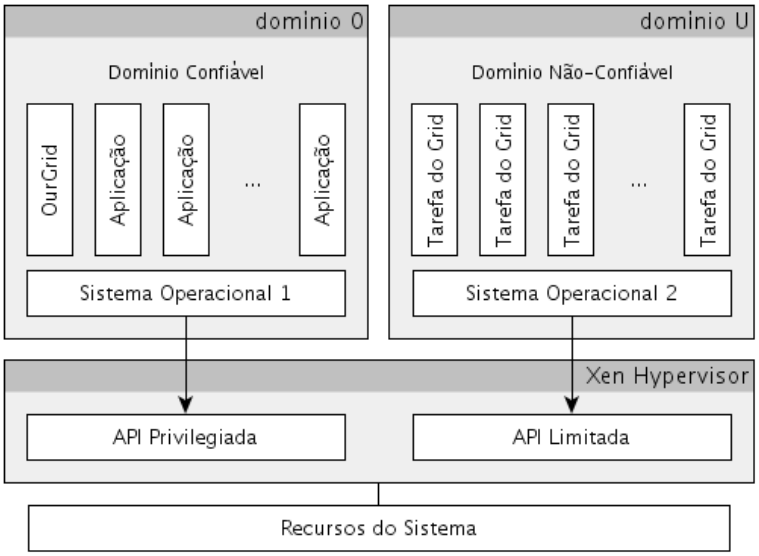

Figura 2. Arquitetura do SWAN Worker

3. SWAN Worker VNET

Se o Job de Eva fosse fortemente acoplado, não seria possível executá-lo da maneira descrita na seção anterior. Nesse caso, seria necessário acrescentar um mecanismo de conexão entre as máquinas usadas por Eva, para que estas formassem uma rede virtual entre si.

3.1. Requisitos

Existem alguns desafios em fornecer para Eva tal ambiente de execução. Abaixo, nós listamos os requisitos que uma solução precisa atender para que o usuário consiga executar, de forma segura, aplicações paralelas em múltiplos sites com comunicação entre suas tarefas:

- **Isolamento**: sendo o OurGrid um grid aberto, é essencial proteger os recursos de usuários maliciosos. Atualmente, o SWAN Worker protege os recursos físicos de aplicações maliciosas (ex. ataques DoS e danos ao sistema de arquivos) colocando estas aplicações dentro de um domínio não-confiável do Xen sem conectividade de rede. Esta solução resolve eficientemente o problema de segurança, mas apenas para aplicações desacopladas. No novo contexto discutido nesse artigo, o mecanismo de segurança não deve apenas confinar a aplicação, mas também o tráfego gerado pela mesma. Para tal é preciso criar uma rede virtual para proteger os recursos físicos de ataques e também ser capaz de executar aplicações paralelas acopladas.

- **Configuração sob demanda**: um dos pilares conceituais do OurGrid é a simplicidade de implantação; dessa forma o mecanismo de implantação da rede virtual deve ser autônomo, não exigindo a realização de quaisquer configurações complexas antes da submissão de um job. Para isso, nós precisamos prover um mecanismo de implantação sob demanda que prepara o ambiente em tempo

de execução sem qualquer envolvimento com os administradores dos recursos.

- **Contornar assimetria da rede**: o mecanismo deve estar habilitado a contornar a assimetria da rede imposta por *firewalls* e NATs. Em outras palavras, o mecanismo não deve assumir que existe conectividade bidirecional entre todos recursos alocados para a aplicação do usuário. Com isso, a rede virtual deve prover conectividade entre todos os recursos alocados para a aplicação, mesmo com a presença *firewalls* e NATs entre estes recursos.

3.2. Solução

Nesta seção nós apresentamos um mecanismo de segurança que atende aos requisitos descritos na seção anterior. A solução proposta consiste em um padrão de endereçamento e um mecanismo de implantação do Xen VNET.

Xen VNET é um mecanismo que permite a conexão de todos os domínios não-confiáveis dentro de uma rede virtual que é completamente isolada do "mundo exterior" [Wray 2006, Kallahalla et al. 2004]. O Xen VNET provê isolamento encapsulando todo tráfego gerado na rede virtual dentro de pacotes EtherIP [Housley and Hollenbeck 2002] identificados e separados por VNET ID (seqüência de 128 bits) que são transportados pela rede física. A comunicação entre os recursos que estão em uma mesma rede virtual é feita através de um canal *multicast* ou através de um canal *unicast* UDP. O Xen VNET pode ser utilizado ou como um módulo do kernel ou como um *daemon*, as duas formas devem ser executadas no domínio confiável. A Figura 3 mostra a arquitetura do OurGrid com Xen VNET no caso de uso discutido anteriormente. De acordo com a Figura 3, podemos observar que cada site possui sua própria VNET (VNET A, VNET B e VNET C) e cada *OG Peer* está dentro de duas VNETs, uma VNET do seu próprio site e outra VNET para comunicação entre todos os *OG Peers* (VNET D). Note que a VNET D somente é criada a partir do momento em que a aplicação submetida recebe recursos de outros sites.

3.2.1. Endereçamento

A especificação do endereçamento é dividida em duas partes: VNET ID e atribuição de IP. Primeiramente nós apresentamos a convenção que utilizamos para identificar uma VNET e logo em seguida nós explicamos nossa convenção para atribuição de IPs.

Um requisito fundamental da nossa solução é prover isolamento entre o tráfego gerado por aplicações distintas. Dessa forma, é esperado que o tráfego gerado por um Job A não seja "visto" por um Job B e vice-versa. Então, para garantir isolamento entre as aplicações, nós definimos que haverá uma relação um-para-um entre aplicação e topologia virtual. Para conseguir tal relação é necessário identificar unicamente cada VNET que conectará todos os recursos usados por uma aplicação.

No OurGrid, cada aplicação submetida possui uma identidade de requisição denominada de Request Id. Cada requisição é identificada unicamente por um número aleatório de 64 bits, isto é o bastante para garantir que os primeiros 64 bits de uma VNET ID têm alta probabiblidade de serem globalmente únicos. Este encapsulamento permite

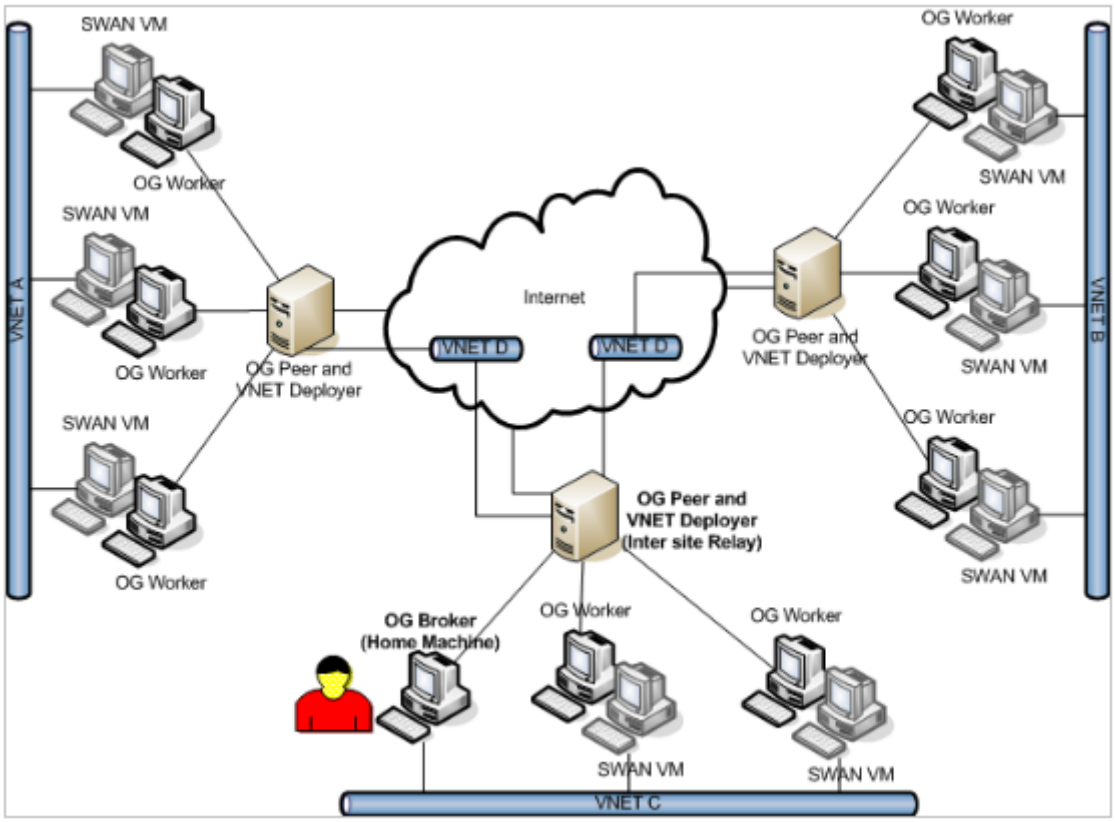

Figura 3. Solução OurGrid com Xen VNET

que cada topologia virtual use qualquer intervalo de IP sem nenhum risco de colisão no espaço de endereços.

Na segunda parte da especificação de endereçamento, é necessário definir como endereços IP são atribuídos aos recursos, os quais fazem parte da rede virtual. Uma escolha natural para este mapeamento seria configurar um servidor DHCP para atribuir IPs para todos os recursos. Entretanto, neste primeiro protótipo decidimos por uma estratégia simples, onde cada recurso determina seu próprio endereço IP baseado em informações locais. Esta abordagem simplifica o design da solução, pois não depende de um componente externo ao OurGrid.

Um conjunto de regras foram definidas para atribuir endereços IP para entidades em uma rede virtual de uma aplicação. Deste modo, o endereço IP atribuído para as interfaces virtuais dos *OG Peers*, *OG Brokers* e *SWAN Workers* de cada aplicação obedece o esquema listado na Tabela 1, onde $SITE\_ID$ é um identificador único de 16 bits para cada site participante da comunidade OurGrid, e $TASK\_ID$ é um número de 16 bits que identifica uma tarefa de uma determinada aplicação.

3.2.2. Mecanismo de implantação da VNET

Após especificar o esquema de endereçamento que atende ao primeiro requisito (isolamento), é necessário construir um mecanismo que configure a conexão de cada recurso

Entidade	Formato do IP
OG Broker	$SITE\_ID.255.252$
OG Peer	$SITE\_ID.255.253$
OG Worker	$SITE\_ID.TASK\_ID$

Tabela 1. IPs Reservados

com a rede virtual. A idéia é ter um mecanismo totalmente distribuído onde cada recurso efetua sua própria configuração sob demanda.

Para implantar uma rede virtual agregando todos os recursos alocados para uma determinada aplicação, configurações automáticas precisam ser feitas em todos os sites que doam recursos para esta aplicação. Estas configurações são descritas a seguir:

Criando canais UDP entre recursos. Para que ocorra comunicação entre os recursos, o Xen VNET requer que todos estes recursos façam parte do mesmo canal multicast. Visto que esta configuração é inviável, uma opção é a criação de canais UDP em ambas direções entre os recursos, de forma que a topologia virtual se assemelhe a um clique. Entretanto, essa solução requer que os recursos possuem conectividade bidirecional, o que não é a norma atualmente devido a existência de firewalls e NATs. Além disso, mesmo que a conectividade bidirecional fosse a norma, manter uma topologia completamente conectada não é escalável. Sendo assim, os *OG Peers* se tornam recursos estratégicos na topologia virtual servindo como relay entre *OG Brokers*, *OG Workers* e outros *OG Peers*, mantendo canais UDP com estes recursos. Os *OG Peers* ainda configuram tabelas de roteamento com o objetivo de fazer com que *OG Brokers* e *OG Workers* se comuniquem.

Criando e configurando interfaces virtuais. Interfaces virtuais precisam ser criadas na máquina do usuário, em todos os recursos alocados para uma aplicação, e em cada *OG Peer* que está provendo recursos para esta aplicação em particular. Como é esperado que o *OG Worker* execute apenas uma tarefa por vez, então é necessário somente ter uma interface virtual. *OG Brokers* e *OG Peers* terão uma interface virtual por aplicação. Todos os recursos vão configurar suas respectivas interfaces virtuais independentemente, de acordo com a convenção do endereçamento IP descrita anteriormente.

Atualizando tabelas de roteamento (se necessário). Finalmente, *OG Peers* precisam configurar suas tabelas de roteamento para controlar propriamente o tráfego de chegada e saída. Ajustando corretamente a tabela de roteamento, todos os recursos poderão se comunicar dentro da topologia virtual. A configuração do roteamento é necessária para resolver o problema de não se ter conectividade bidirecional entre todos os recursos.

Voltando ao caso de uso da cientista Eva, temos que Eva submete um job para o *OG Broker Eva* que envia uma requisição pedindo recursos para o *OG Peer A*. O *OG Broker Eva* ainda cria um canal UDP partindo dele mesmo até o *OG Peer A* e uma interface de rede virtual propriamente configurada, isto é, uma interface que estará na mesma rede virtual do *OG Peer A*. Após receber a requisição, o *OG Peer A* cria e configura uma nova interface de rede virtual, atualiza sua tabela de roteamento e redireciona a requisição para

a comunidade. Se o *OG Peer A* possui recursos locais ociosos que satisfazem os requisitos da requisição, então novos canais UDP são criados entre o *OG Peer A* e estes recursos. Da mesma forma, se o *OG Peer A* receber uma resposta da comunidade, novos canais UDP serão criados entre o *OG Peer A* e todos os outros peers que estão doando recursos. Além disso, todas as rotas serão adicionadas na tabela de roteamento do *OG Peer A*, permitindo a comunicação com estes recursos remotos. Note que os peers remotos terão uma configuração similar à configuração do *OG Peer A* para cada requisição atendida.

Na visão do processo de implantação do SWAN VNET, os passos para a submissão de um Job são descritos logo a seguir. Note que a implantação é feita de forma distribuída, onde cada componente da arquitetura configura uma parte.

1. O *OG Broker Eva* cria uma interface virtual que está na rede virtual com o VNET ID que foi gerado baseado no Request ID do job;
2. O *OG Broker Eva* cria um canal UDP para o *OG Peer A*;
3. O *OG Peer A* cria uma interface virtual com o mesmo VNET ID do *OG Broker Eva*;
4. O *OG Peer A* cria canais UDP para todos os seus *OG Workers* ociosos que satisfazem os requisitos do job;
5. O *OG Peer A* configura sua tabela de roteamento, roteando todos pacotes vindos dos *OG Workers* para o *OG Broker Eva* e vice versa;
6. Cada *OG Worker* requisitado cria uma interface virtual com o mesmo VNET ID do *OG Peer A* e *OG Broker Eva*.
7. Se algum *OG Worker* for recebido da comunidade, o *OG Peer A* cria um canal UDP para o *OG Peer* dono deste *OG Worker*
8. O *OG Peer A* configura sua tabela de roteamento, roteando todos pacotes vindos dos *OG Workers* vindos da comunidade para o *OG Broker Eva* e os *OG Workers* do *OG Peer A* e vice versa;
9. O mesmo procedimento de criação de interfaces virtuais e canais UDP, bem como as configurações de rotas, é feito para cada *OG Peer* que doar recursos para outro *OG Peer*.

4. Trabalhos Relacionados

Existem vários artigos que discutem os problemas e sugerem soluções para a execução de aplicações paralelas através de múltiplos sites. Entretanto, nenhum deles oferece uma solução baseada em um grid aberto que provê a implantação de uma rede virtual sob demanda.

Nós dividimos os trabalhos relacionados em 4 classes distintas: solução específica de aplicação [Allen et al. 2001], infraestrutura [de Jong and Koot 2006], bibliotecas [Foster and Karonis 1998, Kielmann et al. 1999], e redes virtuais [Tatezono et al. 2006, Wolinsky et al. 2006, Ganguly et al. 2006, Jiang and Xu 2003].

Em [Allen et al. 2001] é proposta uma solução específica de aplicação para agregar quatro supercomputadores. Eles obtiveram êxito agregando recursos através de múltiplos sites com o objetivo de executar aplicações paralelas. Os autores conseguiram executar uma aplicação que resolveu um problema cinco vezes maior do que outros problemas resolvidos anteriormente. Dada a natureza específica da aplicação, o trabalho

não foca no problema de controlar a assimetria imposta por NATs e *firewalls*. Eles controlaram isso envolvendo administradores de todos os sites que participaram do experimento.

Um desafio relacionado com execução de aplicações paralelas envolvendo múltiplos sites é contornar as limitações de latência e incompatibilidade de implementações MPI. Em [de Jong and Koot 2006] são discutidas as dificuldades para se ter um ambiente de execução MPI composto por recursos de um conjunto de domínios administrativos independentes. Os conflitos de configuração entre os sites é um dos principais problemas descritos neste trabalho.

Em [Foster and Karonis 1998] é descrito e avaliado o MPICH-G, uma biblioteca para infraestrutura de grids que esconde a complexidade de diferentes bibliotecas de comunicação enquanto mantém a mesma interface exportada pelo ambiente de execução original do MPI (ex. mpirun, machines file). Uma solução similar, mas um pouco mais específica, é descrita em [Kielmann et al. 1999].

O trabalho apresentado em [Wolinsky et al. 2006] descreve e avalia um sistema que provê a implantação dinâmica de máquinas virtuais com o objetivo de executar computação com alta vazão em um ambiente largamente distribuído. Esta solução combina a utilização de máquinas virtuais com uma rede overlay que atravessa *firewalls* e NATs para permitir a criação de Condor-based pools onde os nós são estações de trabalho virtuais (VMware ou Xen). Ao contrário deste trabalho, a nossa solução é voltada para um grid aberto onde o risco de se ter aplicações maliciosas é muito maior, tornando claro o motivo para o uso de mecanismos de proteção para os recursos usados na execução da aplicação, incluindo o isolamento do tráfego gerado pela aplicação dentro de uma rede virtual.

Em [Jiang and Xu 2003] os autores apresentam o VIOLIN, uma arquitetura de redes virtuais que, diferentemente do nosso trabalho, deve ser implantado em uma infraestrutura overlay (ex. PlanetLab). O VIOLIN utiliza a tecnologia de máquinas virtuais User Mode Linux (UML) [Dike 2001] e além de nós fim virtuais, o VIOLIN também virtualiza nós que funcionam como roteadores ou switchs.

A rede virtual chamada IP-over-P2P [Ganguly et al. 2006] permite comunicação através de *firewalls* e NATs entre suas máquinas virtuais. Diferentemente do nosso trabalho, onde o mecanismo de implantação da rede virtual usa uma convenção própria para determinar os IPs, eles usam servidores DHCP para distribuir os IPs. Como esta rede virtual é uma rede fechada, torna-se mais fácil a implantação de servidores DHCP para dar nomes e IPs automaticamente para os recursos.

5. Conclusão

Neste artigo, nós apresentamos uma arquitetura para a execução de aplicações paralelas acopladas em um grid aberto e seguro, com recursos localizados em diferentes domínios administrativos. Para que esse tipo de aplicação execute nesse ambiente, é necessário que todos os recursos que estão executando uma aplicação possam se comunicar. Para isto, nós estendemos a arquitetura do OurGrid para permitir a execução de aplicações acopladas agregando recursos de diferentes domínios administrativos sem no entanto comprometer as premissas de segurança e facilidade de uso. Com esta arquitetura, será possível executar aplicações paralelas através de vários sites lidando com várias limitações e aten-

dendo aos requisitos de isolamento, envolvimento administrativo mínimo e tratamento de assimetria da rede.

A nossa solução ainda está em fase de implementação, porém, nós já temos um protótipo onde conseguimos executar uma aplicação sintética escrita na linguagem Java e que utiliza RMI (*Remote Method Invocation*) para a comunicação entre seus processos. Ainda tentamos executar neste protótipo, aplicações MPI reais (e.g. BRAMS [INPE/CPTEC 2007] e mpiBLAST [Darling et al. 2003]), mas tivemos problemas de *timeout* da conexão entre os processos.

Como trabalhos futuros, após o termino da implementação, nós vamos fazer uma avaliação medindo o *overhead* imposto por nossa solução executando aplicações paralelas (e.g. MPI). Note que, o experimento na ausência da nossa solução deve ser feito envolvendo os administradores de cada site, onde devem ser configurados *firewalls* e/ou túneis com o objetivo de deixar todos os recursos "visíveis" entre si. Ainda pretendemos implementar um mecanismo de servidores DHCP para controlar automaticamente os nomes e IPs para os recursos.

Agradecimentos

Este trabalho foi desenvolvido em colaboração com a HP Brasil P&D.

Referências

(2007). Sandbox. `http://en.wikipedia.org/wiki/Sandbox\_\(computer\_security\)`.

Allen, G., Dramlitsch, T., Foster, I., Goodale, T., Karonis, N., Ripeanu, M., Seidel, E., and Toonen, B. (2001). Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus. In *Supercomputing Conference*.

Andrade, N., Brasileiro, F., and Cirne, W. (2004). Discouraging free-riding in a peer-to-peer grid. In *Thirteenth IEEE International Symposium on High-Performance Distributed Computing*.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauery, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. In *19th ACM Symposium on Operating Systems Principles*.

Cirne, W., Brasileiro, F., Andrade, N., Costa, L., Andrade, A., Novaes, R., and Mowbray, M. (2006). Labs of the world, unite!!! *Journal of Grid Computing*, 4(3).

Darling, A., Carey, L., and Feng, W. (2003). The design, implementation, and evaluation of mpiblast. In *4th International Conference on Linux Clusters: The HPC Revolution*.

de Jong, R. and Koot, M. (2006). Preparing the worldwide lhc computing grid for mpi applications. `http://staff.science.uva.nl/~delaat/snb-2005-2006/p35/presentation.pdf`.

Dike, J. (2001). User mode linux (uml). `http://user-mode-linux.sourceforge.net/als2001/index.html`.

Foster, I. and Karonis, N. (1998). A grid-enable mpi: message passing in heterogeneous distributed computing systems. In *Supercomputing Conference*.

Ganguly, A., Agrawal, A., Boykin, O. P., and Figueiredo, R. (2006). IP over P2P: Enabling self-configuring virtual ip networks for grid computing. In *IPDPS*.

Housley, R. and Hollenbeck, S. (2002). Etherip: Tunneling ethernet frames in ip datagrams. RFC 3378.

INPE/CPTEC (2007). Brazilian regional atmospheric modeling system (brams). `http://www.cptec.inpe.br/brams/`.

Jiang, X. and Xu, D. (2003). VIOLIN: Virtual internetworking on overlay infrastructure. In *Department of Computer Sciences Technical Report CSD TR 03-027 - Purdue University*.

Kallahalla, M., Uysal, M., Swaminathan, R., Lowell, D. E., Wray, M., Christian, T., Edwards, N., Dalton, C. I., and Gittler, F. (2004). SoftUDC: A software-based data center for utility computing. *Computer*, 37(11).

Kielmann, T., Hofman, R. F. H., Bal, H. E., Plaat, A., and Bhoedjang, R. A. F. (1999). Magpie: Mpi's collective communication operations for clustered wide area systems. In *Symposium on Principles and practice of parallel programming*.

Santos, R., Andrade, A., Cirne, W., Brasileiro, F., and Andrade, N. (2005). Accurate autonomous accounting in peer-to-peer grids. In *3rd Workshop on Middleware for Grid Computing*.

Tatezono, M., Maruyama, N., and Matsuoka, S. (2006). Making wide-area, multi-site mpi feasible using xen vm. In *Workshop on XEN in High-Performance Cluster and Grid Computing Environments*.

Wolinsky, D., Agrawal, A., Boykin, O., Davis, J., Paramygin, V., Sheng, P., and Figueiredo, R. (2006). On the design of virtual machine sandboxes for distributed computing in WOWs. In *1st International Workshop on Virtualization Technology in Distributed Computing*.

Wray, M. (2006). Vnet - domain virtual networking. `http://www.cl.cam.ac.uk/research/srg/netos/xen/`.