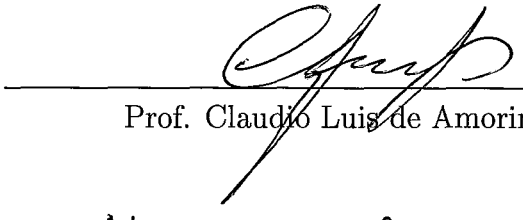
MIGRAÇÃO DE MÁQUINAS VIRTUAIS EM AMBIENTES DE GRADES COMPUTACIONAIS

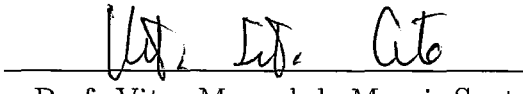
Marconi Féo Pereira Rivello de Carvalho

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

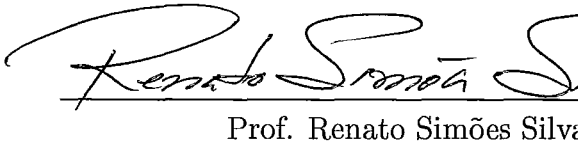
Aprovada por:



Prof. Claudio Luis de Amorim, Ph.D.



Prof. Vitor Manuel de Morais Santos Costa, Ph.D.



Prof. Renato Simões Silva, Dr.

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2006

CARVALHO, MARCONI FÉO PEREIRA RIVELLO DE

Migração de Máquinas Virtuais em Ambientes de Grades Computacionais [Rio de Janeiro] 2006

XI, 46 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2006)

Dissertação – Universidade Federal do Rio de Janeiro, COPPE

1 - Migração de Máquinas Virtuais

2 - Virtualização

3 - Grades Computacionais

I. COPPE/UFRJ II. Título (série)

Aos meus pais...

Agradecimentos

A Deus, por me dar capacidade;

Ao prof. Renato Simões Silva, por todo o incentivo, oportunidades, e valiosas lições ensinadas no LNCC;

Ao Lauro Whately, pela preciosa co-orientação não-oficial;

Ao prof. Claudio Luis de Amorim, pela orientação e disponibilização de recursos no LCP;

Aos meus pais, pelo insubstituível exemplo a seguir;

Ao Eduardo Melione, pelo companheirismo desde a graduação;

À Aline Brandão e à Sermograf, pela impressão e encadernação desta dissertação;

Aos colegas e professores da COPPE, pelo auxílio e prestatividade nos momentos necessários;

Ao CNPq, pelo essencial apoio financeiro.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

MIGRAÇÃO DE MÁQUINAS VIRTUAIS EM AMBIENTES DE GRADES COMPUTACIONAIS

Marconi Féo Pereira Rivello de Carvalho

Setembro/2006

Orientador: Claudio Luis de Amorim

Programa: Engenharia de Sistemas e Computação

Esta dissertação apresenta uma proposta de utilização de máquinas virtuais em *grids*, para execução de aplicações científicas, com o objetivo de diminuir as restrições impostas pelo *software* neles instalado, e de prover novas facilidades, antes inexistentes.

Foram desenvolvidos e implementados mecanismos para migrar máquinas virtuais entre máquinas pertencentes ao *grid*, o que possibilita a implementação de técnicas de balanceamento dinâmico de carga e tolerância a falhas, além da simples facilidade de mover aplicações em execução entre máquinas. Os mecanismos desenvolvidos têm por objetivo não somente realizar a migração, mas realizá-la de forma eficiente. Tal objetivo é alcançado explorando as características dos *grids*, dos Sistemas Operacionais e dos sistemas de arquivos usados atualmente.

Abstract of Dissertation presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

VIRTUAL MACHINES MIGRATION IN GRID ENVIRONMENTS

Marconi Fêo Pereira Rivello de Carvalho

Setembro/2006

Advisor: Claudio Luis de Amorim

Department: Systems Engineering and Computer Science

In this dissertation, we propose the use of virtual machines to run CPU-bound scientific applications in Grid Environments so that the Grid's native software becomes less restrictive to the applications while virtualization provides new features unavailable previously.

In support to this proposal, we developed and implemented new mechanisms to allow migration of virtual machines in Grids. Besides enabling running applications to be transferred between Grid nodes when necessary we noticed that those mechanisms in turn allowed for dynamic load balancing and fault tolerance techniques to be easily implemented in Grids.

Our experimental results showed that the mechanisms we proposed not only could afford the transfers of virtual machines suitably but also efficiently in Grids. To do so, the mechanisms exploited carefully the main characteristics of Grids, operating systems, and file systems that are used currently.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Computadores pessoais se tornaram uma ferramenta comum a todos os ramos da ciência, indústria e comércio, são dotados de um alto poder de processamento e usados para os mais diversos fins e aplicações. Algumas vezes, tenta-se extrair o máximo de desempenho na realização de tarefas computacionalmente intensivas e exigentes de recursos computacionais, enquanto que em outras, o *hardware* é sub-utilizado.

Em grandes instituições, são encontrados computadores sub-utilizados e usuários demandando mais recursos computacionais. O objetivo das *grids*[25] é disponibilizar os recursos ociosos para quem precisa, podendo a *grid* limitar-se a uma instituição, ou ser estendida a outras instituições que estejam dispostas a colaborar.

Para gerenciar os recursos e usuários, utiliza-se um *grid middleware*, como MyGrid[28] ou Globus Toolkit[24, 23], que é uma camada de *software*, entre o sistema operacional e as aplicações, que fornece a abstração de *grid* e permite que um usuário que deseja utilizar um recurso computacional especifique seus pré-requisitos, como CPU, RAM e disco disponíveis, bem como sistema operacional e outros pré-requisitos de *software*. Um *grid middleware* também é responsável pela autenticação dos usuários, descobrimento e gerência de recursos, além da submissão dos *jobs* e coleta de resultados.

O uso de grids geograficamente distribuídos é uma realidade, tanto no Brasil quanto mundialmente. Instituições cooperam entre si, fornecendo recursos ociosos umas às outras, em troca da disponibilidade de recursos quando necessário. Quando a barreira inter-instituição é ultrapassada, aumenta-se a preocupação com segurança. Disponibilizar *hardware* ocioso para pessoas que estejam precisando é uma atitude nobre, mas não se deve pôr em risco o sistema que receberá a aplicação convidada.

Por isso, é imprescindível haver isolamento entre os sistemas convidados e locais.

Outro problema que surge pela utilização de máquinas de terceiros é a falta de controle sobre o sistema. Alguém que empresta uma máquina para outra pessoa executar uma aplicação não vai querer que seu sistema seja alterado, o que deixa o usuário convidado sem muita liberdade sobre o ambiente de execução de sua aplicação. Há casos em que para a aplicação ser executada corretamente deve ser feita alguma modificação no sistema, como a instalação de uma biblioteca, ou a substituição por uma versão específica de bibliotecas. Nestes casos, o usuário remoto deverá procurar outra máquina hospedeira para sua aplicação.

Como o controle sobre a máquina é local, o responsável pela máquina pode a qualquer momento precisar utilizá-la, movê-la para outro local, realizar algum procedimento de manutenção, ou, até por um descuido, reiniciar o sistema sem checar se há usuários remotos utilizando-a. Em qualquer desses casos, o processamento realizado seria perdido e o usuário remoto teria que iniciar novamente seu processo em outra máquina hospedeira. A possibilidade de interromper o processamento e continuá-lo em outra máquina, fazendo a migração da aplicação, seria altamente desejável, além de permitir a implementação de balanceamento de carga através da migração de aplicações.

Para realizar migração, pode-se usar *checkpointing*. *Checkpointing* é uma técnica que consiste em, de tempos em tempos, salvar o estado da aplicação para ser utilizado posteriormente, caso seja necessário. Técnicas similares[32, 12, 29, 33] são utilizadas com eficácia em sistemas homogêneos, como *clusters* de PCs, mas não são adequadas a *grids*, devido à grande heterogeneidade tanto de *hardware* quanto de *software*.

O problema da migração de processos é a dificuldade de se identificar o escopo dos mesmos. Processos não têm um escopo bem definido, podendo, a qualquer momento, utilizar novos arquivos, abrir novas conexões de rede, ou requisitar novos recursos, impedindo definir o que deve ou não ser migrado juntamente com o processo.

Devido ao aumento de camadas de *software*, o código do usuário passa a não mais depender apenas do *hardware* disponível, mas também do sistema operacional, das bibliotecas de sistema e dos compiladores disponíveis. Dessa forma, mesmo havendo o *hardware* disponível para o usuário, não será possível executar sua aplicação caso não haja um sistema operacional específico instalado, ou de alguma biblioteca da qual sua aplicação dependa.

Grids são uma solução prática, viável e eficiente em muitos casos, como na

execução de aplicações do tipo *bag-of-tasks*[1], mas ainda há espaço para melhorias, como vimos acima.

Visando remover os problemas descritos acima, nesta dissertação propomos que os usuários não mais submetam aplicações diretamente ao *grid*, mas máquinas virtuais contendo todo o conjunto de *software* necessário ou mais adequado à sua aplicação em especial. Mais ainda, a submissão de aplicações através de máquinas virtuais deverá ser transparente, mantendo o atual processo de submissão de aplicações à *grid*.

Há diversos tipos e implementações de máquinas virtuais. Para nossa finalidade, precisamos de máquinas virtuais que permitam a execução de sistemas operacionais convencionais, mantendo a ABI (*application binary interface*) compatível com as aplicações existentes, para minimizar o trabalho na adoção do novo paradigma.

Esta proposta traz consigo soluções e novos problemas. Como solução imediata, temos a eliminação das dependências de *software*. Como não existe "almoço grátis", temos como contrapeso o *overhead* da camada adicional introduzida na arquitetura – a saber, o monitor de máquinas virtuais (*virtual machine monitor* - VMM) – e o *software* adicional submetido juntamente com a aplicação do usuário (a máquina virtual).

Além do benefício citado acima, o uso da virtualização permitirá um melhor isolamento entre os usuários da *grid*, além de permitir a implementação de técnicas de migração, que podem ser usadas para balanceamento de carga e tolerância a falhas.

Para validar a proposta, medimos a sobrecarga introduzida pela virtualização e, como pode ser verificado na seção de experimentos, concluímos que a de processamento é irrisória. Com relação à sobrecarga da migração, propomos, implementamos e avaliamos técnicas para minimizá-la, em diferentes cenários.

O processo de migração se dá explorando a estrutura física dos sistemas de arquivos[35, 14, 18, 5, 19], a preocupação dos desenvolvedores de sistemas operacionais em manter a compatibilidade com versões anteriores de sistemas operacionais e também o *software* encontrado comumente em diferentes distribuições Linux.

Uma máquina virtual submetida ao *grid* pode conter programas, bibliotecas, ou outros tipos de *software* que também se encontram na máquina física de destino, mas com diferenças necessárias ao usuário. Nesse caso, é desejável que se transfira

[1]Aplicações paralelas cujas tarefas são independentes.

apenas a parte que não se encontra na máquina de destino[30], que é a forma como resolvemos o problema da migração.

Em resumo, as contribuições desta dissertação são:

- Estudo da utilização de máquinas virtuais em ambientes de grades computacionais: a virtualização permitirá remover as restrições de *software*, proverá isolamento entre as aplicações dos usuários e permitirá a implementação da migração que, por sua vez, possibilitará a implementação de técnicas de balanceamento de carga e tolerância a falhas;

- Proposta e implementação de técnicas para minimizar a sobrecarga da migração;

- Avaliação da sobrecarga introduzida pela virtualização, para aplicações científicas CPU *bound*;

- Avaliação do sistema implementado, em diferentes cenários de (*grids*).

Dividimos a dissertação da seguinte forma: no capítulo 2, fazemos uma breve descrição de máquinas virtuais, suas aplicações e principais formas de implementação. Descrevemos a utilidade de máquinas virtuais em ambientes de grades computacionais no capítulo 3. No capítulo 4, descrevemos o sistema experimental e sua implementação. Damos prosseguimento, no capítulo 5, aos experimentos e resultados obtidos. No capítulo 6, expomos os trabalhos relacionados, e apresentamos nossas conclusões no capítulo 7.

Capítulo 2

Máquinas Virtuais

Uma máquina virtual é um ambiente criado em *software*, que simula o comportamento de uma plataforma computacional. A máquina real, que está executando a virtual, é chamada de hospedeira, por fornecer os recursos de *hardware* necessários à execução da máquina convidada (virtual).

Assim como modelos computacionais procuram reproduzir o mundo real, as máquinas virtuais reproduzem outras máquinas reais. Devido ao aperfeiçoamento das tecnologias de processadores e memória, tal reprodução não possui mais restrições computacionais. É possível executar um computador em outro, sendo este igual ou completamente diferente do hospedeiro.

A seguir, são apresentadas algumas aplicações e técnicas de implementação de máquinas virtuais.

2.1 Motivação

2.1.1 Emulação

É muito comum um usuário possuir um certo *hardware* e querer executar *software* desenvolvido para outra arquitetura de *hardware*. Com computadores de uso geral, como PCs, é possível desenvolver um programa que se comporte como a arquitetura desejada. Esses programas são chamados de emuladores, e têm por objetivo executar software desenvolvido para uma determinada arquitetura.

Normalmente, se utiliza emuladores de arquiteturas diferentes da máquina real. Mas nada impede que seja desenvolvido um emulador para executar na arquitetura que se deseja emular[1]. Tal uso não é muito comum, por existirem técnicas mais eficientes de simular o comportamento da arquitetura da máquina real, aproveitando recursos do próprio *hardware*.

Existem diversos emuladores disponíveis na *internet* para *download*. Os mais comuns são os que emulam *video-games* e computadores antigos. Com eles, é possível executar jogos e programas que não estão disponíveis para máquinas recentes.

2.1.2 Simulação

Um simulador visa reproduzir com fidelidade todos os aspectos da arquitetura alvo. Não basta que o resultado seja o mesmo, os mecanismos que levam ao resultado também devem ser os mesmos. O objetivo é, além de reproduzir o resultado obtido com a execução do binário, reproduzir o comportamento interno da arquitetura.

É muito útil para estudo de arquitetura de computadores, para propostas de modificações em arquiteturas existente, ou propostas de novas arquiteturas, podendo estimar o desempenho, sem a necessidade de se criar um protótipo real do *hardware*.

2.1.3 Execução de ambientes diferentes

Uma necessidade que se torna cada vez mais significativa é a execução de sistemas operacionais diferentes em uma mesma máquina. Muitos profissionais e estudantes precisam utilizar mais de um sistema operacional, ou sistemas operacionais iguais configurados de forma diferente, ou com programas e bibliotecas de versões diferentes. Como é financeiramente inviável alocar uma máquina física para cada sistema operacional, estes usuários acabam possuindo apenas uma máquina. A solução mais comum para este problema é o *dual-boot* (dupla inicialização do sistema), ou seja, possuir os dois ou mais sistemas operacionais instalados no disco da máquina, e selecionar no momento da inicialização qual deve ser ativado. Essa solução é satisfatória quando não há a necessidade de se alternar entre os sistemas operacionais freqüentemente. Quando houver tal necessidade, por exemplo, no desenvolvimento de aplicações multi-plataforma, é desejável executar os dois sistemas operacionais concorrentemente, o que é impossível utilizando somente os recursos fornecidos pelo *hardware* de PCs.

2.1.4 Experimentos de Risco

Para profissionais e pesquisadores da computação, é comum surgir a necessidade de se testar um software ou procedimento que, se não correr tudo como planejado, podem causar danos irreparáveis ao sistema ou aos dados armazenados. Uma solução

pouco prática é fazer *backup* antes dos experimentos. Ao invés disso, uma máquina virtual poderia ser usada, já que destruindo-a não acarretará nenhum dano real.

Como exemplo de aplicação, podemos citar *honeypots*. *Honeypots* são máquinas deixadas propositalmente como alvo para ataques em uma rede, com o objetivo de detectar tentativas de invasão ou de acesso não-autorizado à rede. Em vez de se usar uma máquina real, pode-se alocar uma máquina virtual para executar o *honeypot*.

2.1.5 Rede

Em aplicações de rede, há a necessidade de se ter disponíveis múltiplas máquinas inter-conectadas. Muitas das vezes, isso não é possível por falta de recursos, ou não é possível em tempo hábil. A solução é executar diversas máquinas virtuais, em uma mesma máquina física, provendo também uma forma de conexão entre elas.

Uma vantagem muito grande de se utilizar máquinas virtuais para esse fim, é a facilidade de criação e alteração da topologia da rede virtual, já que não há necessidade de cabos e equipamentos reais para se montar uma rede extremamente complexa.

2.1.6 Proteção entre usuários

Em ambientes multi-usuários surge a necessidade de se modificar o sistema, ou adicionar bibliotecas e programas diferentes, que podem interferir no funcionamento de outros programas ou usuários. Em alguns casos, utiliza-se máquinas separadas, o que acaba sendo desperdício de recursos.

Fornecer uma máquina virtual a cada usuário é uma forma simples de atender a necessidades diferentes, e às vezes mutuamente exclusivas. Cada usuário, então, pode administrar seu sistema virtual por completo, modificando o que bem entender, sem prejudicar aos outros usuários e seus programas, nem ao sistema hospedeiro. O usuário passa a ser o administrador de sua máquina virtual, não precisando pedir permissão a ninguém, nem aguardar por alterações no sistema.

2.2 Técnicas para desenvolvimento de máquinas virtuais

Existem diferentes formas de se desenvolver uma máquina virtual. Cada uma com seus prós e contras, com suas peculiaridades e objetivos.

2.2.1 Emulação

Emulação[1, 13] consiste em reproduzir o funcionamento de uma arquitetura (alvo), normalmente, utilizando outra (hospedeira). Um software interpretador recebe como entrada um binário desenvolvido para a arquitetura alvo, e o interpreta, reproduzindo o comportamento esperado utilizando o *hardware* da arquitetura hospedeira.

Como o *hardware* alvo não está disponível, deve ser feita uma tradução das instruções da arquitetura alvo para as da hospedeira. O que normalmente ocorre, é que para executar uma instrução da arquitetura alvo, são necessárias diversas instruções da hospedeira, por não haver uma equivalência direta no conjunto de instruções, causando assim, uma (normalmente grande) degradação de desempenho.

Essa técnica é muito utilizada quando se deseja emular uma arquitetura ultrapassada em máquinas recentes. Nesses casos, o desempenho não se torna um problema.

2.2.2 Virtualização

A técnica de virtualização[9, 13, 4] consiste em fornecer um ambiente idêntico ao da máquina real. Por isso, é possível executar partes do binário desejado diretamente em *hardware*, sem a necessidade de interpretação por *software*. Em máquinas x86, isso não pode ser feito para todas as instruções.

As partes do binário que podem ser executadas diretamente pela CPU, são executadas na mesma velocidade (ignorando efeitos de *cache*, esvaziamento de *pipeline*, e outros detalhes de otimização) do *hardware* real. As partes que não podem ser executadas, como instruções privilegiadas, devem ser tratadas pelo monitor da máquina virtual (VMM).

Sistemas operacionais desenvolvidos para a máquina hospedeira podem ser executados pela máquina virtual sem qualquer modificação.

No caso da arquitetura x86, a virtualização é uma tarefa muito complexa, pois instruções privilegiadas, quando executadas em modo usuário, falham silenciosamente. Para resolver esse problema, utiliza-se uma técnica chamada *binary rewriting*, que consiste em localizar as instruções privilegiadas e substituí-las por *traps* que serão interpretadas pelo monitor de máquinas virtuais.

2.2.3 Para-virtualização

Para evitar alguns problemas da virtualização, a para-virtualização[37, 21] visa fornecer um *hardware* virtual com algumas diferenças do real. Essas modificações podem visar menor complexidade de implementação, ou maior desempenho.

Por haver diferenças entre a arquitetura real e a virtual, sistemas operacionais desenvolvidos para a arquitetura hospedeira precisam ser alterados para que seja possível sua execução na máquina virtual. Dependendo das modificações feitas na arquitetura virtual, as alterações necessárias no sistema operacional convidado podem ser mínimas, ou impraticáveis, sendo no caso da última, necessário escrever um novo sistema operacional para a arquitetura virtual implementada.

2.2.4 Execução em modo usuário

Um sistema operacional pode ser modificado para ser executado em modo usuário[7, 13]. Dessa forma, ele passa a ser executado como um processo dentro do sistema operacional hospedeiro. São necessárias modificações no código que acessa recursos do hardware, e que utiliza instruções privilegiadas, para que passem a utilizar recursos fornecidos pelo sistema operacional hospedeiro, através de *system calls*.

Dessa forma, diversas máquinas virtuais podem ser executadas simplesmente rodando o executável do sistema operacional modificado para modo usuário.

Pode-se fazer modificações no sistema operacional hospedeiro, para facilitar a implementação ou para melhorar o desempenho. Mas não são necessárias.

2.3 Xen: Monitor de Máquinas Virtuais

Xen [21] é um MMV baseado em paravirtualização para a arquitetura IA-32. Xen pode, de forma segura, executar múltiplas MVs em um único computador com desempenho próximo do hardware real[15].

Xen [1], atualmente, pode ser executado na arquitetura IA-32, especificamente em qualquer processador "P6" ou mais novo, tais como Pentium Pro, Celeron, Pentium II, Pentium III, Pentium IV, Xeon, AMD Athlon e AMD Duron.

Desde a versão 3 do Xen, a arquitetura X86/64 (Opterons e Xeon/64) é suportada, enquanto que o suporte para a arquitetura IA64 (Itanium) está em de-

[1]Especificamente, Xen é o nome dado à implementação do VMM, mas geralmente se refere a todo sistema composto pelo VMM, máquinas virtuais e todo o conjunto de *software* de controle e gerenciamento relacionados.

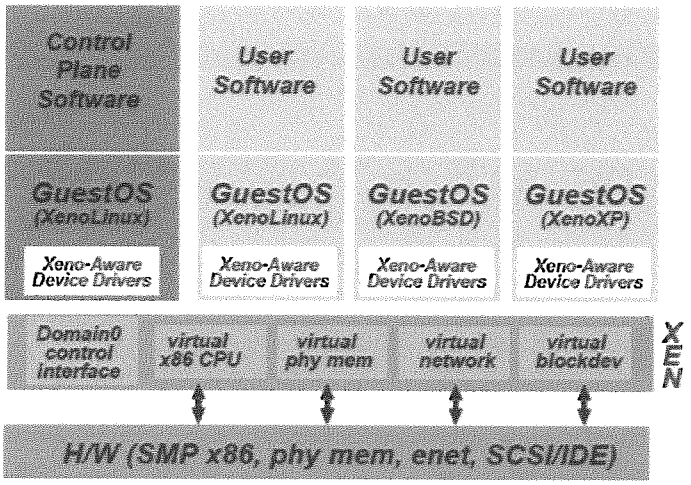


Figura 2.1: Monitor de Máquinas Virtuais Xen

senvolvimento. Máquinas multiprocessadas são reconhecidas pelo MMV e os SOs convidados podem utilizar até 32 processadores. Há, também, suporte para extensão de memória nas máquinas x86 (*Physical Address Extension - PAE*, permitindo o uso de até 64GB de memória física.

Em um sistema Xen para IA-32, o MMV é executado no anel 0. Os SOs convidados devem usar os anéis 1,2 ou 3. É esperado que o SO use o anel 1 e coloque as aplicações no anel 3. O MMV tem acesso a toda a memória física disponível e é responsável pela alocação de partes da memória para os domínios. A segmentação da memória é usada para impedir que o SO convidado acesse o espaço de endereços reservado para o MMV.

2.3.1 Interação entre o MMV e os Domínios

Em geral, as MVs (domínios, na terminologia Xen) são executados pelo MMV da mesma forma em que processos são executados por um SO. Eles podem ser suspensos e reativados transparentemente e podem invocar serviços do MMV através das *hypercalls*. Diferente de uma chamada de sistema de um SO, uma *hypercall* não pode bloquear e deve ser vista como um mecanismo de notificação. A comunicação do MMV para o domínio toma a forma de "eventos" que substitui os mecanismos de interrupção e exceções do processador. Existem alguns tipos de eventos, cada um indicando um tipo particular de ocorrência, por exemplo a chegada de um pacote da rede, ou que o pedido de leitura do disco completou. Como a comunicação

entre o MMV e os SOs convidados é apenas usada para notificações, o MMV é completamente assíncrono.

Além de exportar instâncias virtualizadas do processador, memória, rede e discos, Xen oferece também uma interface de controle para gerenciar os recursos que são compartilhados entre os domínios. O acesso à interface de controle é restrito a uma MV com privilégios especiais, conhecida como "domínio 0". Ele pode criar e remover outros domínios, controlar os parâmetros de escalonamento do processador e a alocação de recursos, entre outras tarefas. O domínio 0 é parte necessária a todo servidor Xen, que ao executar o *software* de controle, separa a execução do mecanismo e da política dentro do sistema.

2.3.2 Processador e Temporizadores

Virtualizar o processador envolve prover abstrações para temporizadores e interrupções, como também prover um mecanismo para o compartilhamento do processador entre os domínios.

Xen provê aos SOs convidados o três tempos: *uptime* real, virtual e o tempo *wall-clock*. O *uptime* real é expresso em nanosegundos e é medido desde que o sistema iniciou. O *uptime* virtual do domínio apenas avança quando ele está sendo executado e é usado para o escalonamento dos processos no SO convidado. O *wall-clock* é especificado como um *offset* para ser adicionado ao *uptime* real, obtendo-se, assim, a hora real. Um cliente NTP pode ser usado no domínio 0 para atualizar o tempo real. Cada domínio recebe um par de temporizadores, para o tempo real e o tempo virtual do domínio. O alarme do tempo virtual pode ser usado para o escalonamento dos processos no SO convidado, enquanto os outros componentes podem usar o temporizador do tempo real. Os alarmes são sinalizados através do mecanismo de evento.

O VMM pode controlar o número de eventos gerados pelos dispositivos e enviados para os domínios. Por exemplo, um SO pode pedir para receber um evento apenas após um número n de pacotes ter sido recebido e estar pronto para ser entregue, ou após decorrido um tempo t desde a chegada do primeiro pacote. O controle de eventos permite trabalhar com requerimentos de latência e vazão de um domínio específico.

Xen oferece três escalonadores:

BVT provê um escalonador do tipo *proportional fair share*. Provê despacho com

baixa latência usando *virtual-time warping*, um mecanismo que temporariamente viola o compartilhamento justo "ideal" em favor de domínios que receberam o evento esperado de um dispositivo.

Atropos pode ser usado para reservar quotas (*shares*) absolutas do processador para cada domínio. Cada domínio tem um período e uma fatia do tempo de processador associados, assim o domínio deve receber uma "fatia" do processador a cada "período" de tempo.

Round Robin é oferecido como um exemplo de implementação de um escalonador Xen, e funciona da mesma forma que o implementado em SOs convencionais.

Os parâmetros do escalonamento, por domínio, podem ser ajustados pelo *software* de controle no domínio 0.

2.3.3 Memória

A alocação da memória inicial, para cada domínio, é especificada no momento da criação. A memória é, então, estaticamente particionada entre os domínios, provendo um forte isolamento. Um tamanho máximo de memória pode ser especificado: se um domínio aumenta a demanda por memória, ele pode tentar pedir outras páginas de memória para o Xen, até o seu limite especificado. Da mesma forma, o domínio pode devolver, para o Xen, páginas de memória que não esteja usando. Existe um módulo no SO convidado conhecido como "*balloon driver*", que ajusta o tamanho da memória física usada no respectivo domínio. Xen controla o "*balloon driver*" para que este reserve páginas virtuais do SO convidado e as mantenha "presas" às respectivas páginas físicas. As páginas físicas usadas podem, então, ser retiradas do domínio pelo Xen.

OS SOs convidados são responsáveis pela alocação e gerenciamento das tabelas de páginas no *hardware*, com o mínimo de envolvimento do Xen, que fiscaliza o uso correto do mapeamento das páginas e segmentos, apenas para assegurar a segurança e o isolamento. Sempre que o SO convidado criar uma nova entrada na tabela de páginas (por exemplo, na criação de um processo), esta deve ser registrada no Xen. Todas as atualizações na tabela de páginas devem ser validadas pelo MMV. Um SO pode apenas mapear páginas que possui e não pode escrever na tabela de páginas. As atualizações são acumuladas para amortizar a sobrecarga da troca do contexto para o MMV.

2.3.4 Dispositivos de E/S

Na sua última versão, Xen implementa a arquitetura *Safe Hardware Interface*, uma arquitetura que permite *drivers* de dispositivos, sem modificação, serem compartilhados por MVs isoladas, enquanto protege cada instância de SO e o sistema como um todo de falhas nos *drivers*. Domínios de dispositivos, na terminologia Xen, são MVs específicas para executar *drivers* de dispositivos. Isso permite a compatibilidade com todos os dispositivos que possuam *drivers* para o SO Linux, por exemplo. Os domínios comuns (chamados de "domínios convidados") executam um *frontend driver* que se comunica através do um "canal do dispositivo", implementado no MMV, com o *backend driver* no domínio de dispositivos. Os domínios de dispositivos acessam diretamente os dispositivos possuídos, entretanto as interrupções vindas dos dispositivos são tratadas primeiro pelo MMV, que notifica o domínio de dispositivos correspondente através do mecanismo de eventos. O domínio convidado troca pedidos de serviço e respostas com o domínio de dispositivos através do anel de descritores de E/S, no canal de dispositivos.

2.3.4.1 Virtualização da Rede

Do ponto de vista dos outros domínios, o *backend driver* de rede no domínio de dispositivos correspondente, implementa um comutador *ethernet* virtual, onde cada domínio pode ter uma ou mais interfaces virtuais conectadas. O *backend driver* é responsável por uma variedade de ações relacionadas à transmissão e recepção dos pacotes do dispositivo real. Com relação à transmissão, as seguintes ações podem ser realizadas:

Validação: o cabeçalho de cada pacote é conferido para assegurar que os endereços MAC e IP são verdadeiros. As funções de validação podem ser configuradas usando iptables, no caso do SO Linux, por exemplo.

Escalonamento: o *backend driver* deve escalonar o acesso dos vários domínios que possuam pacotes na fila de transmissão. A função de escalonamento assume um esquema básico de limitação da taxa de transmissão.

Registro de Eventos e Contabilidade: o *backend driver* pode ser configurado para registrar eventos, tal como a tentativa de um domínio enviar um pacote TCP contendo um SYN.

No recebimento dos pacotes, o *backend* age como um demultiplexador. Os pacotes são encaminhados à interface virtual correspondente, depois de realizados os respectivos registro e contabilidade.

2.3.4.2 Virtualização de dispositivos de bloco

Todos os acessos do SO convidado ao disco são feitos através da interface de dispositivos de bloco virtual (*virtual block device - VBD*). Essa interface permite que os domínios acessem partes dos dispositivos de armazenamento visíveis ao *backend device*. Um anel de descritores, no MMV, compartilhado entre o *frontend* e o *backend devices*, permite a troca de pedidos e respostas de acesso ao disco virtual.

2.3.5 *Live Migration*

Live Migration[16] é usado para transferir um domínio entre diferentes máquinas, sem suspender a execução do SO convidado no domínio. Da perspectiva do usuário, a migração deve ser imperceptível.

Na implementação corrente, não há suporte para a transferência do conteúdo armazenado nos discos locais. Os administradores devem escolher uma solução de armazenamento apropriada (por exemplo, SAN, NAS, etc) para assegurar que o sistema de arquivos no SO convidado também esteja disponível na máquina destino.

Quando um domínio migra, seus endereços MAC e IP também são transferidos. Por esse motivo, só podem ser transferidas as MVs que estejam dentro da mesma camada 2 de rede e sub-rede IP. Se a máquina de destino estiver em uma sub-rede diferente, o administrador precisará configurar manualmente um *etherip*[27] ou um túnel IP no domínio 0 da máquina de destino.

Um domínio pode estar usando muita memória, acarretando, assim, em uma transmissão muito demorada, mesmo em uma rede veloz. Xen, para minimizar o *downtime* da aplicação, mantém o domínio executando normalmente enquanto ocorre a migração, resultando na suspensão da execução do domínio migrado por um tempo entre 60 e 300 ms[16].

Capítulo 3

O Uso da Virtualização em Grades Computacionais

Máquinas Virtuais deixaram de ser apenas uma ferramenta de desenvolvimento e testes, e têm sido usadas em ambientes de produção, para os mais variados fins, graças ao bom desempenho obtido nas atuais implementações dessas máquinas, tais como VMware[9], User-mode Linux[7], QEMU[13], Xen[21], entre outras.

Um exemplo de utilização de máquinas virtuais em ambiente de produção é a "hospedagem"[8] de máquinas virtuais na internet. Ao invés de hospedarem portais, ou qualquer outro serviço, hospedam máquinas virtuais, oferecendo aos clientes controle total sobre sua execução. De posse de uma máquina virtual, o usuário pode instalar e executar o sistema operacional de sua preferência, bem como o *web server* que desejar, ou qualquer outro aplicativo. No capítulo de experimentos, apresentaremos resultados que confirmam a baixa sobrecarga de implementações recentes de máquinas virtuais.

As soluções atuais de *grids* podem se beneficiar enormemente do uso de máquinas virtuais, mas há um preço a ser pago. Como pontos negativos, além da sobrecarga da virtualização, que será vista com mais atenção ao longo desta dissertação, temos:

- Impossibilidade do uso de sistemas operacionais proprietários: este problema é específico dos monitores de máquinas virtuais implementados usando para-virtualização, pois o sistema operacional convidado precisa ser portado para a arquitetura virtual. Este problema pode ser solucionado adotando-se uma implementação que virtualize o *hardware* por completo, como VMware e QEMU, ou com o uso de *hardware* que suporte virtualização nativamente, como as arquiteturas Pacifica[10] e VT[17], da AMD e Intel, respectivamente;

- Necessidade de se ter o monitor de máquinas virtuais instalado nas máquinas da *grid*: um custo administrativo que, dependendo do administrador do sistema, pode ser desprezível ou inviabilizador;

- Suporte dos desenvolvedores: é necessário que o projeto do monitor de máquinas virtuais continue em desenvolvimento. Caso o desenvolvimento seja interrompido, pode-se adotar outro monitor de máquinas virtuais.

No entanto, acreditamos que os benefícios compensam, como veremos a seguir.

3.0.6 Restrições de *Software*

A primeira vantagem, é a liberdade provida ao usuário para escolher seu ambiente de execução. Uma mesma máquina física pode atender a diversos usuários, cada um com necessidades distintas de *software*.

Programas desenvolvidos para sistemas operacionais diferentes, podem ser executados simultaneamente em uma máquina hospedeira de máquinas virtuais (*host*), eliminando este pré-requisito.

Além de restrições quanto ao sistema operacional em si, programas podem depender de versões específicas de um sistema operacional, ou até mesmo de um conjunto de bibliotecas, impossibilitando a execução de outros programas que necessitem de versões diferentes.

Outro problema bem comum é a questão da atualização do sistema operacional. Apesar de ser recomendado por inúmeros motivos, como correções de falhas de segurança e aumento de desempenho, uma atualização pode, ocasionalmente, fazer com que programas previamente instalados produzam resultados indesejáveis. Usando máquinas virtuais, a máquina hospedeira pode estar sempre atualizada, e os programas que apresentarem incompatibilidades podem ser executados em máquinas virtuais com uma versão do sistema operacional mais antiga.

Cada um dos ítens citados acima pode ser facilmente configurado em uma máquina virtual, já que o usuário possui total controle sobre a mesma.

3.0.7 Isolamento

Permitir acesso de terceiros ao sistema pode ser uma atitude arriscada. Apesar dos sistemas operacionais modernos, como Linux, proverem diversos mecanismos

de isolamento entre usuários, sabemos que falhas de segurança são freqüentemente encontradas, e podem comprometer todo o sistema.

Máquinas virtuais fornecem mais uma camada de isolamento, entre o sistema operacional hospedeiro e o convidado, impedindo que o sistema convidado acesse qualquer recurso (que não tenha sido pré-configurado) do sistema hospedeiro.

3.0.8 Tolerância a Falhas

Em um ambiente compartilhado, onde diversos usuários podem estar trabalhando simultaneamente, o estresse do sistema é maior que em um sistema dedicado, e sua complexidade aumenta proporcionalmente ao número de usuários com necessidades diferentes.

O sistema é mais exigido tanto em termos de *hardware* quanto de *software*. Mais usuários significa mais acesso a disco, mais uso de CPU, memória, rede, enfim, todo o *hardware* sofre maior demanda. Da mesma forma, principalmente quando os usuários possuem necessidades diferentes, o *software* também é sobrecarregado. Controlar processos de diversos usuários, bancos de dados, *logs*, sistemas de arquivos, etc, se torna uma tarefa mais complexa, portanto, mais propensa a erros.

O uso de máquinas virtuais permite que esse problema seja abordado de forma simples e eficaz. A cada usuário, ou grupo de usuários com as mesmas necessidades com relação a *software*, é fornecida uma máquina virtual contendo todo o conjunto de *software* necessário aos mesmos. Esta máquina virtual possui seu estado de CPU, memória RAM e disco virtual independente das demais e bem definidos.

A partir deste momento, o ambiente de cada usuário (ou grupo de usuários) estará isolado dos demais, permitindo uma melhor administração e implementação de políticas de tolerância a falhas independentes umas das outras.

Cada máquina virtual em execução pode ter seu estado (CPU, memória RAM e disco virtual) salvo em um disco local ou remoto (esta técnica é conhecida por *checkpointing*), podendo ser restaurado posteriormente em caso de falha, o que permitirá continuar a execução da máquina virtual a partir de um estado consistente, evitando recomeçar todo o trabalho realizado anteriormente.

Checkpointing é apenas um exemplo de técnica de tolerância a falhas que pode ser implementada com o uso de máquinas virtuais. Um ponto interessantíssimo ao se utilizar máquinas virtuais é que cada máquina virtual pode ser "protegida" por uma técnica diferente, não necessitando que usuários com restrições mais ou menos

rigorosas tenham que entrar em comum acordo. Cada um escolhe a abordagem que lhe for mais conveniente ou apropriada.

Uma segunda situação muito delicada é quando há a necessidade de manutenção na máquina física, como troca de disco, *cooler*, mudança física da máquina de local, ou qualquer outra operação que necessite do desligamento da máquina, e a máquina está ocupada com processos de usuários. Vários processos podem já estar sendo executados há dias, por exemplo, portanto, seria desejável que o trabalho até aquele ponto não fosse desperdiçado. Mesmo que os usuários tenham optado por não utilizar métodos de tolerância a falhas, podem aproveitar as características já citadas de máquinas virtuais para salvar o estado de execução de forma que este seja retomado após o procedimento de manutenção.

3.0.9 Migração e Balanceamento de Carga

Em grandes parques computacionais é comum haver a possibilidade de escolha da máquina onde se irá executar um trabalho. Em um determinado momento, a melhor opção disponível pode não ser, efetivamente, a mais apropriada.

Suponhamos o seguinte caso: possuímos duas máquinas: M1 e M2, sendo M1 50% mais rápida que M2. Há, também, dois usuários: U1 e U2, ambos necessitando executar tarefas que demandam dias de computação. O usuário U1 chega primeiro, e inicia seu processo na máquina M1, que levará alguns dias para executar o trabalho. O usuário U2, não tendo outra alternativa, inicia seu trabalho na máquina M2. Após quatro dias de execução, o trabalho do usuário U1 termina, deixando a máquina M1 livre. O usuário U2 estima, pelo andamento de seu processo, que a execução de seu trabalho se encontra aproximadamente na metade. Neste momento, há uma máquina mais rápida disponível. Mas, pelo andamento do processo, não valeria a pena iniciá-lo desde o início na outra máquina. Seria ótimo se fosse possível simplesmente transferir o processo do usuário U2 da máquina M2 para a máquina M1 e retomar sua execução.

Se o trabalho do usuário U2 estiver sendo executado em uma máquina virtual, este pode interromper sua execução, transferir seu estado para a máquina mais rápida e continuar a execução do exato ponto onde parou.

Ao contrário da migração de processos, o estado de máquinas virtuais é bem delimitado, facilitando imensamente o processo de migração.

Aproveitando o exemplo anterior, de tolerância a falhas, caso a manutenção

da máquina física seja demasiado demorada, as máquinas virtuais ativas podem ser migradas para outra máquina física, evitando a interrupção dos trabalhos em execução.

Capítulo 4

Sistema de Transferência de Máquinas Virtuais

Neste capítulo, apresentamos nossa proposta de utilização de máquinas virtuais em ambientes de grades computacionais. São discutidas suas peculiaridades, apresentadas as abordagens escolhidas para lidar com as dificuldades encontradas e descritas as técnicas desenvolvidas e sua implementação.

4.1 Infra-estrutura

O sistema é construído utilizando como base o Xen[21] (monitor de máquinas virtuais), por motivos de possuir o melhor desempenho[21, 15] dentre as implementações disponíveis, estar em um estágio maduro e, ao mesmo tempo, sendo constantemente aprimorado conforme as novas versões liberadas.

O Xen exporta um hardware virtual semelhante – porém não idêntico – à arquitetura x86, exigindo assim, que o sistema operacional a ser executado sobre ele seja modificado. Mas, por as diferenças serem pequenas, as modificações necessárias não são muitas. Basicamente, é necessário substituir instruções privilegiadas por chamadas ao *hypervisor* (o VMM). A ABI (*application binary interface*), interface entre o sistema operacional e as aplicações, não sofre modificações, o que permite que aplicações continuem sendo executadas na nova plataforma sem qualquer modificação.

Cada máquina do grid deve possuir o Xen, que gerenciará as máquinas virtuais em execução na máquina hospedeira. A máquina física é dividida em domínios, cada um com uma parte da memória RAM, disco(s) e interface(s) de rede virtuais. A gerência do sistema é feita a partir do domínio 0, que é inicializado automaticamente.

Por ele, é possível gerenciar os demais domínios e máquinas virtuais.

O sistema operacional a ser executado no domínio 0 é uma modificação do Linux para a arquitetura do Xen. Como aconteceu com o User-mode Linux[7], espera-se que as modificações sejam incorporadas ao fonte oficial do kernel do Linux.

Nos demais domínios, pode-se executar qualquer sistema operacional portado para a arquitetura do Xen (como Linux, NetBSD, Plan9 e FreeBSD).

Com a introdução das CPUs com suporte a virtualização em *hardware* (Intel VT[17] e AMD Pacifica[10]), é possível executar qualquer sistema operacional para a arquitetura x86 sem a necessidade de portá-lo para a arquitetura paravirtualizada do Xen.

4.2 Estrutura dos Discos

Com relação aos discos e sistemas de arquivos, cada máquina do *grid* passará a ter uma partição com o sistema de arquivos para o sistema operacional hospedeiro, e uma área disponível para alocar os sistemas operacionais convidados.

Dependendo dos recursos disponíveis na máquina física, pode-se utilizá-la para hospedar apenas uma única máquina virtual por vez (que teria à disposição quase todos os recursos de *hardware* da máquina física, excetuando-se apenas o necessário ao VMM), ou diversas máquinas virtuais simultaneamente.

Caso se deseje executar apenas uma máquina virtual por vez, pode-se reservar uma única partição no disco físico, que será destinada a alocar o sistema de arquivos convidado. Esta abordagem é bastante simples, e não exige esforços por parte do administrador do sistema.

Se o objetivo for executar concorrentemente duas ou mais máquinas virtuais em uma mesma máquina hospedeira, a alocação de espaço em disco já não é trivial. A alocação e desalocação de espaço para máquinas virtuais, que se iniciam e terminam em momentos distintos, pode ocasionar a fragmentação do espaço livre em disco, impedindo que o mesmo seja alocado para uso em uma máquina virtual.

Uma alternativa a criar e remover partições convencionais no disco é armazenar os sistemas de arquivos convidados em forma de arquivos convencionais no sistema operacional hospedeiro. Esses arquivos podem, então, ser utilizados pelas máquinas virtuais como *virtual block devices*. Apesar de muito mais conveniente, o uso de *file-backed file systems*[34] pode acarretar em maior *overhead*, dependendo da utilização

que a máquina virtual faz do disco, além da posição em que os blocos do disco virtual forem armazenados no disco real, pois os sistemas operacionais otimizam o acesso seqüencial ao disco. Arquivos muito grandes podem ser alocados mais ou menos continuamente no disco, ou completamente fragmentados. Isso será feito de acordo com a disposição dos blocos livres no disco. Um sistema de arquivos "velho"[1] tem menos chances de oferecer grandes áreas contíguas para armazenar os arquivos que um sistema de arquivos recém criado e populado. Como os sistemas operacionais procuram ordenar o acesso a disco de modo a obter melhor desempenho considerando a linearidade do acesso, um sistema de arquivos *file-backed*, caso esteja muito fragmentado, pode reduzir consideravelmente o desempenho do sistema como um todo.

Uma melhor abordagem com um bom comprometimento entre praticidade e desempenho, é o uso do *logic volume manager* (LVM)[3]. Com o LVM, podemos utilizar uma grande partição no disco físico, ou mesmo todo um disco à parte, para armazenar os sistemas de arquivos convidados de forma simples e sem grandes perdas de desempenho. Esta área é chamada de *volume group*. O LVM divide a área alocada em blocos de mesmo tamanho, chamados de *physical extents* (PE). Da mesma forma como um sistema de arquivos utiliza diversos blocos para alocar os arquivos, o LVM utiliza diversos *physical extents* para alocar um *logical volume* (LV). Cada *logical volume* será visto pelo sistema operacional como uma partição, e poderá ser formatado e populado com um sistema de arquivos convidado. O LVM procura alocar os PEs de forma contínua, para obter um melhor desempenho, mas quando não é possível, a alocação pode ser feita utilizando PEs descontínuos, que apesar de não ser tão eficiente, é eficaz. A figura 4.1 mostra uma visão geral do LVM.

4.3 Transferência dos Discos Virtuais

Os dados em disco são a parte mais crítica da migração, portanto, merecem atenção especial e são o foco principal deste trabalho, que será discutido nesta seção.

Um fator que aumenta muito a complexidade da manipulação dos sistemas de arquivos é a grande quantidade de variações encontradas. Foi-se o tempo em que os dados eram atrelados ao código. E, da mesma forma que os programas se separaram

[1] Que sofreu muitas gravações e exclusões, principalmente de arquivos muito grandes.

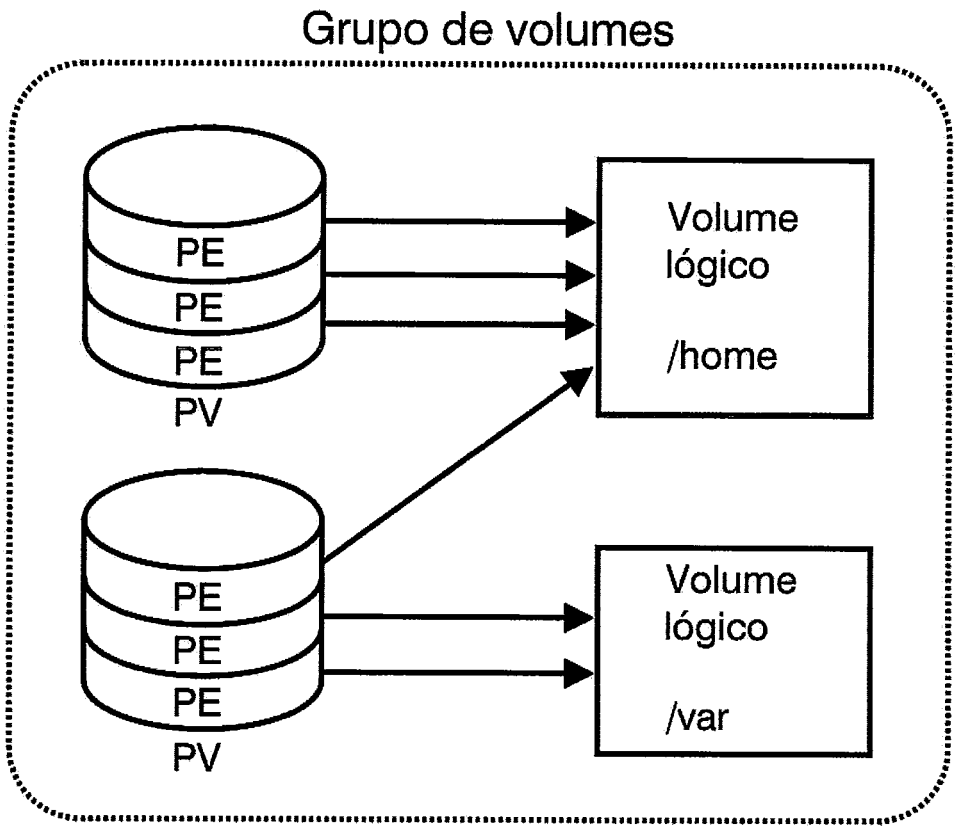


Figura 4.1: Visão geral do LVM

das bases de dados, permitindo diversas combinações de SGBDs[2] e interfaces de usuário, os sistemas operacionais se separaram dos sistemas de arquivos.

Há inúmeros sistemas de arquivos *open source* e comerciais disponíveis para um mesmo sistema operacional. Por exemplo: para Linux, os mais comuns atualmente são o *ext3* e o *ReiserFS*, enquanto que para WindowsXP, há a opção de se utilizar *FAT32* ou *NTFS*. Cada um com suas características e estruturas de dados próprias. Isso, sem levar em consideração possíveis extensões, o que torna ainda mais complexa a manipulação dos sistemas de arquivos.

Uma característica que todos os sistemas de arquivos compartilham é da divisão do mesmo em blocos. Um dos motivos de o fazer, é a limitação de endereçamento. Ao invés de endereçar *bytes*, endereça-se blocos, permitindo o uso de maior espaço de armazenamento com o mesmo tamanho de endereço. Ao identificar o tamanho do bloco utilizado pelo sistema de arquivos, a complexidade de manipulação do mesmo é reduzida drasticamente.

[2]Software Gerenciador de Banco de Dados.

O que torna os dados em disco a parte mais crítica do sistema é a imensa variedade de programas e bibliotecas disponíveis para os usuários. Essa variedade é justamente um dos principais atrativos do uso de máquinas virtuais, pois uma padronização total não pode agradar a todos. No entanto, os sistemas instalados costumam possuir similaridades significativas uns com os outros, mesmo que não sejam completamente compatíveis.

Uma forma de se migrar os dados do disco é ler o sistema de arquivos (para tanto, é necessário conhecer sua estrutura), verificar quais arquivos são encontrados na máquina de destino e transferir apenas os que faltam. Esta abordagem possui algumas desvantagens, uma delas é a necessidade de se conhecer a estrutura e organização dos vários sistemas de arquivos atuais.

Outra forma, é aproveitar o fato dos sistemas de arquivos serem divididos em blocos de tamanho fixo, usualmente 4KB, e dividir os arquivos em pedaços de igual tamanho para armazená-los em disco. Assim, podemos comparar blocos de origem com blocos de destino, evitando a complexidade de se conhecer a estrutura do sistema de arquivos, bastando saber apenas o tamanho de bloco que o sistema utiliza.

Para comparar os blocos de destino com os de origem, de forma eficiente, podemos obter uma assinatura digital (*hash*) dos blocos, e compará-las, ao invés de comparar os dados em si. Optamos pelo algoritmo SHA1, que gera *hash* de 20 *bytes*. Considerando o tamanho dos blocos de 4KB, o *overhead* criado pelo *hash* será de menos de 0,5% do tamanho do disco da máquina virtual, possibilitando uma economia muito maior do volume de dados a ser transferido efetivamente.

4.3.0.1 Explorando a Similaridade dos Sistemas

O uso da técnica de *hash* permitirá a identificação eficaz dos "trechos" de um sistema de arquivos pertencentes também a outro.

Ao aplicar essa técnica à migração de máquinas virtuais, conseguimos evitar a transferência de arquivos, ou partes de arquivos, já existentes localmente, sem a necessidade de se conhecer por completo a estrutura dos sistemas de arquivos envolvidos, mesmo que sejam completamente diferentes, um do outro.

Foram desenvolvidos um servidor e um cliente para transferência de sistemas de arquivos, que fazem uso de *hashing* para evitar transferência desnecessária de dados.

O servidor tem por objetivo disponibilizar um sistema de arquivos qualquer, que pode ser usado em uma máquina virtual. O cliente, conecta-se ao servidor, e

requisita o envio do *hash*, para então requisitar apenas os blocos que não podem ser encontrados no sistema de arquivos local.

Os sistemas de arquivos são divididos em blocos de tamanho fixo, determinado no momento da formatação da partição. Os blocos são alocados para armazenar informações sobre o próprio sistema de arquivos (meta-dados), como número total de blocos e blocos disponíveis, e para armazenar os arquivos e diretórios criados pelos usuários (os dados, em si).

Independente do sistema de arquivos utilizado, cada arquivo do sistema será dividido em blocos e, normalmente, é armazenado de forma descontínua. O tamanho padrão usado atualmente para os blocos é de 4 *kilobytes*.

Essas características serão exploradas nesse trabalho, para realizar a transferência dos sistemas de arquivos de forma eficiente. Passaremos a enxergar os sistemas de arquivos como um conjunto de blocos de tamanho fixo, pois assim o são. Esta forma de visualização é mais eficiente, para nosso propósito, do que enxergá-los como um conjunto de *bytes*, que é muito simplista e ineficiente, ou de arquivos e diretórios, sendo esta última muito complexa e específica de cada sistema de arquivos.

Um arquivo de 8Kb seria armazenado da mesma forma (de acordo com a nossa visão do sistema de arquivos) em diversos tipos de sistemas de arquivos, como ext2[35] (por conseqüência, ext3), reiserfs[14], fat32[18], e ntfs[5, 19]. Ou seja, seria dividido em dois blocos de 4Kb cada.

Cada sistema de arquivos identifica seu conteúdo utilizando um sistema próprio. Por exemplo, o ext2 identifica os arquivos através de estruturas denominadas *inodes*. Então, devemos utilizar algum meio para identificar os diversos blocos contidos nos sistemas de arquivos que serão manipulados.

Para tal, usaremos *hashing*. Uma função de *hash* tem como entrada um número arbitrário de *bytes*, e gera como saída um número fixo de *bytes*. A saída da função de *hash* pode ser usada como uma "assinatura digital". Existem diversos algoritmos de *hash*, dos quais, muitos são usados para fins criptográficos. Foi escolhida a SHA-1, por ser uma das mais seguras funções de *hash*, e por não demandar muito tempo de computação. Por ser computacionalmente inviável encontrar dois conjuntos de entrada diferentes entre si que possuam o mesmo *hash*[26], podemos utilizar *hashing* como forma de identificação dos diversos blocos pertencentes aos sistemas de arquivos.

4.4 Migração das Máquinas Virtuais

O contexto de uma máquina virtual é constituído de seu estado (registradores da CPU), conteúdo da memória principal (RAM) e conteúdo da memória secundária (discos). Desde o início de sua execução, até sua finalização, serão esses os recursos necessários à máquina virtual, ao contrário de processos[32] que, de maneira geral, durante sua execução, podem alocar ou desalocar recursos (arquivos, *sockets*, etc) de forma a impossibilitar a delimitação precisa de seu contexto.

A migração de máquinas virtuais consiste em:

- Interromper a execução da máquina virtual, para que se obtenha um estado consistente;
- Transferir a CPU, memória RAM e disco virtuais;
- Iniciar uma nova máquina virtual na máquina de destino, e restaurar o estado da máquina virtual migrada.

O primeiro e o último ítens são realizados de forma trivial, utilizando procedimentos do monitor de máquinas virtuais (*Virtual Machine Monitor - VMM*).

Não há qualquer dificuldade na transferência da CPU virtual, por se tratar de uma quantidade desprezível de dados, principalmente se comparada ao tamanho da memória e do disco virtuais.

O problema, propriamente dito, se inicia na transferência da memória RAM, e culmina na transmissão do disco virtual.

A quantidade de dados a ser transferida pode variar bastante, de acordo com cada caso. Um sistema operacional pode ser executado em uma máquina (real, ou virtual) com 8 ou 16 *megabytes* de memória, ao passo que outro pode exigir *gigabytes* de memória para sua execução.

Da mesma forma, varia o tamanho do disco virtual. Um sistema mínimo pode utilizar apenas alguns *megabytes* de espaço em disco, enquanto outro pode exigir dezenas ou centenas de *gigabytes*.

Atualmente, o espaço em disco necessário para uma instalação Linux suficiente para ser usada em uma máquina de um *grid*, é de aproximadamente 600 *megabytes*, podendo chegar a algo em torno de 3 *gigabytes* (uma instalação "exagerada" para um *grid*).

Em um ambiente de *grid* comum, um usuário não pode assumir que será possível utilizar toda a memória da máquina, por ser um ambiente compartilhado. Da mesma forma, uma máquina virtual (no caso comum) irá alocar apenas uma parte da memória RAM da máquina real, permitindo que outras máquinas virtuais sejam instanciadas.

Sendo assim, o disco se torna a parte mais crítica da migração, por ser (na grande maioria dos casos) muito maior que a memória RAM.

4.5 Implementação

Para o ambiente experimental, foi necessário o desenvolvimento das aplicações que serão descritas a seguir:

4.5.1 Ferramentas de *hashing*

Foram desenvolvidas ferramentas para gerar os arquivos de *hash* e índice, que serão necessários ao *software* cliente e servidor.

A ferramenta *shafs* toma como entrada uma imagem de um disco virtual, e produz um arquivo contendo o *hash* de cada bloco do mesmo. Este arquivo será utilizado pelo *software* servidor de discos, que o enviará ao cliente, para que o mesmo identifique blocos idênticos aos disponíveis localmente.

A ferramenta *mkidx* utiliza o arquivo de *hash* gerado anteriormente, e gera um outro de índice, ordenado pelo *hash*, para agilizar a pesquisa, e contendo a posição no disco virtual do bloco correspondente. O *software* cliente o usará para acelerar a pesquisa por blocos locais.

4.5.2 Servidor e cliente de discos virtuais

O servidor de discos virtuais utiliza um arquivo de imagem de um disco virtual, e seu arquivo de *hash* correspondente. Ao receber uma conexão TCP, envia os *hashes* do disco provido, e aguarda por requisições (de blocos não encontrados localmente) do cliente. Caso o cliente não encontre algum bloco localmente, prosseguirá verificando quantos dos blocos seguintes também não são encontrados, e faz uma requisição contendo a posição do primeiro bloco não encontrado, e o número de blocos a serem transferidos, a fim de minimizar a latência da comunicação. O servidor, então, lê os blocos requisitados do disco e os envia ao cliente. Após a transferência, o servidor

aguarda por mais requisições, ou a informação de término da migração por parte do cliente. O Algoritmo 1 descreve o funcionamento do *software* servidor.

Algoritmo 1 *Software* Servidor

```
Aguarda conexão
Open(hash)
Open(dados)
Send(hash)
Receive(inicio , count)
while inicio ≠ −1 e count ≠ −1 do
  FSeek(dados , inicio)
  for i ← 1 to count do
    bloco ← FRead(dados)
    Send(bloco)
  end for
  Receive(inicio , count)
end while
```

O *software* cliente conecta-se ao servidor, e usa como entrada um arquivo de imagem de disco virtual local e seu respectivo arquivo de índice. Após a conexão, recebe os *hashes* do servidor, e pesquisa em seu arquivo de índice por blocos com o mesmo *hash*. Caso o bloco seja encontrado, o cliente realiza uma cópia local do bloco, caso contrário, verifica quantos mais, em seqüência, necessita e faz a requisição ao servidor. O comportamento do *software* cliente é descrito no Algoritmo 2.

Versões modificadas foram feitas utilizando bibliotecas de compressão Zlib[20] e LZO[31], com a única diferença de compactar os blocos de dados enviados do servidor ao cliente.

4.5.3 Versão iterativa

A fim de se oferecer um procedimento equivalente ao *live migration*[16] para o disco virtual, foi implementada uma versão iterativa do *software* cliente e servidor.

O funcionamento difere da versão original do cliente e servidor no seguinte aspecto: ao invés do servidor utilizar um arquivo de *hash* previamente gerado, é feita a varredura do arquivo de imagem do disco virtual, calculando-se e armazenando-se em memória o *hash* de cada bloco do disco. Durante esta primeira iteração, os blocos de dados são transferidos ao cliente. Na segunda iteração, a máquina virtual é interrompida para evitar novas alterações no disco virtual, o *hash* é calculado novamente e enviado ao cliente que, dessa vez, requisita apenas os blocos cujo *hash*

Algoritmo 2 *Software* Cliente

```
Conecta ao servidor
Open(index)
Open(disco_in)
Open(disco_out)
Receive(hash)
inicio ← count ← 0
num_blocos ← Conta(hash)
for i ← 1 to num_blocos do
  if Find(hash[i]) then
    if count > 0 then
      Send(inicio, count)
      for j ← 0 to count do
        Receive(bloco)
        Write(disco_out, bloco)
      end for
      count ← 0
    end if
    FRead(disco_in, bloco)
    FWrite(disco_out, bloco)
  else
    if count = 0 then
      inicio ← i
    end if
    count ← count + 1
  end if
end for
if count > 0 then
  Send(inicio, count)
  for j ← 0 to count do
    Receive(bloco)
    Write(disco_out, bloco)
  end for
end if
Send(-1, -1)
```

difere dos da iteração anterior, determinando, assim, quais foram modificados entre a primeira e segunda iteração. Os blocos modificados são, então, transferidos.

Esta versão é utilizada para minimizar o *downtime* da aplicação do usuário, podendo ser usada para migrar o disco virtual contendo os dados do usuário, sem a necessidade de interromper o processamento.

Capítulo 5

Metodologia Experimental

Neste capítulo, temos a parte experimental do estudo. Esperamos avaliar a sobrecarga causada pelo monitor de máquinas virtuais; demonstrar a aplicabilidade da implementação, verificando a presença de blocos idênticos na estrutura física de discos de diferentes instalações Linux; e quantificar a sobrecarga causada pela migração dos discos virtuais.

5.1 Ambiente de Testes

Para avaliar o sistema, utilizamos 3 versões da distribuição Slackware (9, 9.1 e 10), mantendo a mesma seleção de pacotes. As partições foram formatadas com blocos de 4KB (o tamanho padrão). Após a instalação, foram redimensionadas para eliminar todo o espaço não utilizado, e foram geradas imagens em arquivo, por motivos de praticidade para a realização dos experimentos. O tamanho das imagens de cada uma das três versões foi de 706MB, 910MB e 990MB, respectivamente.

Para cada imagem, foi gerado um arquivo contendo o *hash* SHA1[6] de cada bloco da imagem. A partir do arquivo de *hash*, foi gerado um de índice, ordenado pelo *hash*, contendo (além do *hash*) o índice do bloco onde se encontra a primeira ocorrência do *hash* no arquivo de imagem, pois podem haver blocos idênticos em um mesmo sistema de arquivos.

O arquivo de *hash* é uma representação do sistema de arquivos, contendo, ao invés dos blocos de dados de 4096 bytes, blocos de assinaturas digitais de 20 bytes, e será enviado pelo *software* servidor de imagens (que pode estar tanto na mesma máquina física que a máquina virtual a ser migrada, quanto em outra máquina qualquer que possua a mesma imagem do sistema utilizado) ao *software* cliente, ou seja, a máquina de destino da migração. A transferência do *hash* acarreta em um

overhead de menos de 0,5% (20/4096) na quantidade de dados a ser transmitida.

O *software* cliente fará uso do arquivo de índice das imagens disponíveis localmente, para buscar mais eficientemente por um bloco local idêntico a um bloco remoto, e requisitará os blocos não encontrados.

Foram feitas 3 variações na implementação do *software* cliente e servidor: a primeira utiliza-se apenas dos *hashes* e índices para evitar a transferência de blocos desnecessários; a segunda e a terceira são similares à primeira, exceto por compactar os blocos requisitados utilizando, uma a Zlib[20], e a outra a LZO[31].

As máquinas utilizadas nos experimentos foram duas Pentium 4 de 2.8Ghz, com 1GB de RAM, ambas com o sistema operacional Linux 2.6, interconectadas via Ethernet de 100Mbps.

A implementação de máquina virtual adotada foi a Xen 2.0.7.

Para avaliação do sistema, consideramos os seguintes pontos:

- Sobrecarga causada pelo monitor de máquinas virtuais;
- Análise de semelhança entre instalações distintas de distribuições Linux;
- Migração do sistema operacional convidado.

5.2 Sobrecarga do VMM

O objetivo deste experimento é quantificar a sobrecarga introduzida pela paravirtualização, na execução de aplicações científicas, tipicamente CPU *bound.*

O experimento consistiu em executar os *benchmarks* seriais do pacote NAS Parallel Benchmarks[11], versão 3.2, por refletirem o comportamento de aplicações científicas reais, sobre o Linux (sem uso de virtualização), e sobre o XenLinux (versão do *kernel* do Linux modificada para o Xen). Cada *benchmark* foi executado três vezes, e calculada a média.

Na figura 5.1, podemos observar que a diferença de desempenho entre a execução dos *benchmarks* sobre o Linux nativo fica em 1.73%. Considerando que o tempo total de execução foi de aproximadamente 9 horas, 1.73% é uma variação aceitável. O fato surpreendente foi o tempo de execução dos *benchmarks* ter sido menor no XenLinux que no Linux nativo. Apesar da diferença ser pequena, os experimentos foram repetidos e o sistema checado novamente por diferenças que pudessem interferir nos

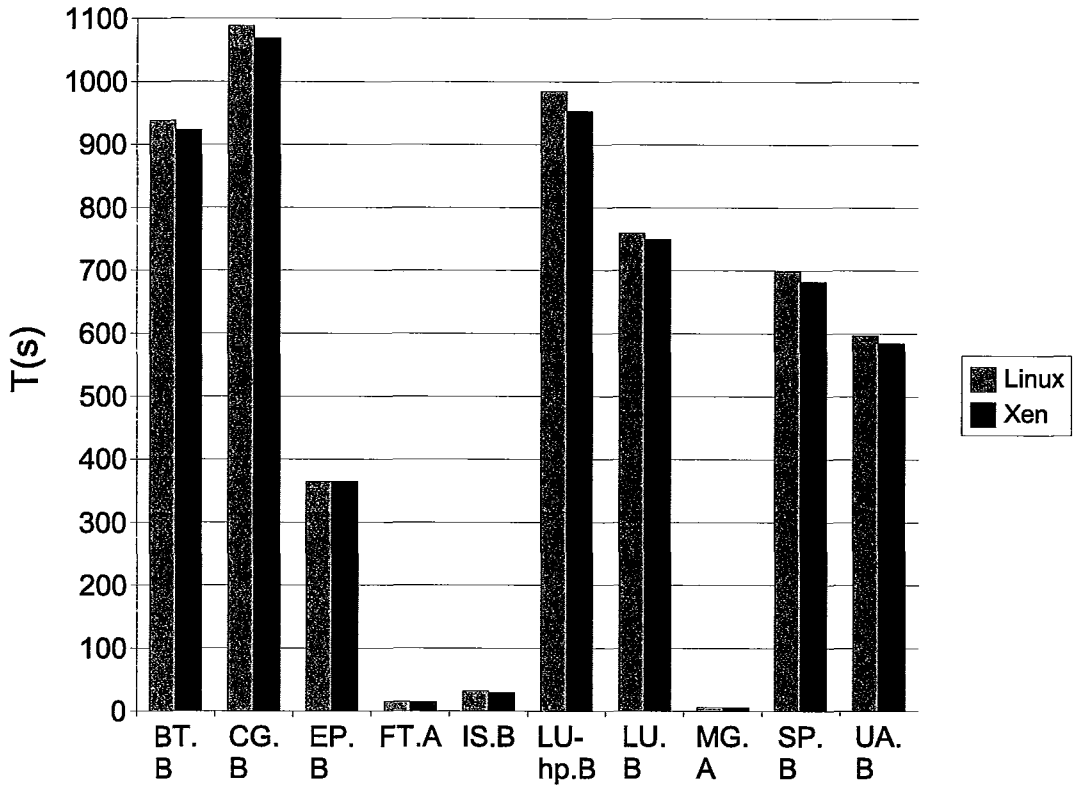


Figura 5.1: Tempo comparativo entre Linux e Xen

resultados (*swap*, *daemons*, etc). Os resultados obtidos foram consistentes com os anteriores.

Sob a ótica deste trabalho, a anomalia encontrada não acarreta maiores conseqüências (além da óbvia admiração), pois o que se desejava demonstrar com os experimentos era a baixa sobrecarga do Xen para aplicações científicas *CPU-bound*, o que pôde ser observado. A explicação mais provável para a anomalia é algum efeito de *cache*, necessitando maiores averiguações que fogem ao escopo deste trabalho.

5.3 Similaridades em Instalações Linux

O sucesso do emprego da técnica descrita depende da quantidade de informação idêntica encontrada simultaneamente em diferentes instalações de distribuições Linux. Sabe-se que é necessário manter a compatibilidade com distribuições mais antigas. Por isso, uma parte significativa da distribuição é mantida ao se fazer uma atualização.

A relevância deste experimento para o trabalho é que ao migrar uma máquina

virtual será feita uma busca na máquina remota por partes em comum do sistema de arquivos a ser transferido, sendo necessário transferir apenas a parte que não possa ser encontrada remotamente.

A distribuição escolhida foi a Slackware, e suas versões 9.0, 9.1 e 10.0, possuem o intervalo entre as *releases* de 6 meses (9.0 para 9.1) e 9 meses (9.1 para 10.0).

OS Convidado	OS Remoto	Similaridade
9.1	9.0	47%
9.1	10.0	34%
10.0	9.0	24%
10.0	9.1	35%
9.1	9.0 e 10.0	51%

Tabela 5.1: Similaridade entre versões diferentes da distribuição Slackware

A tabela 5.1 mostra que, caso haja uma outra instalação Linux na máquina remota, mesmo que contendo versões diferentes de aplicativos e bibliotecas, pode-se aproveitar uma boa parte do sistema disponível remotamente para remontar o sistema convidado, evitando transferir dados desnecessários.

O experimento em que são utilizadas duas distribuições (uma mais antiga, e uma mais recente) para remontar outra mostrou mais de 50% de economia na transferência do sistema operacional convidado.

Os resultados mostram que mesmo que não haja uma instalação idêntica na máquina remota da distribuição utilizada, pode-se utilizar outras disponíveis e remontar parte do sistema operacional convidado.

5.4 Migração

A avaliação da sobrecarga da migração consistiu em transferir, efetivamente, um disco virtual contendo a instalação de uma distribuição Linux para outra máquina, conectada através de uma rede Fast-Ethernet de 100Mbps, ponto-a-ponto.

A avaliação foi dividida em três partes: na primeira, toda a banda estava disponível (100Mbps); enquanto que na segunda, para simular a banda disponível entre sites de um grid, a banda foi restrita em 2Mbps, utilizando o *Linux traffic shaper*[2], por fim, é feita uma transferência real entre o Laboratório Nacional de Computação Científica (LNCC, localizado em Petrópolis, RJ) e o Laboratório de Computação Paralela (LCP, localizado na UFRJ, Rio de Janeiro, RJ). De acordo com os cálculos

feitos levando em consideração a transferência simples, a largura de banda disponível durante a realização dos experimentos[1] foi de, aproximadamente, 20Mbps.

Nas três partes, será migrada a versão 10.0 da distribuição Slackware (no total de 989MB), utilizando as seguintes técnicas:

- Simples - Os dados são transferidos sem qualquer modificação, da mesma forma como é feita uma transferência via FTP.

- Hash - O *hash* do disco é transferido primeiro, e a máquina remota o utiliza para montar o disco virtual utilizando blocos de uma instalação do Slackware 9.1, disponível localmente. Os blocos que não foram encontrados localmente foram requisitados e transferidos sem modificação.

- Hash-Zlib - A transferência foi feita da mesma forma como no ítem acima, com a única diferença de ser aplicada compressão utilizando a biblioteca Zlib nos blocos que foram requisitados pela máquina remota.

- Hash-LZO - Foi feita da mesma forma como no ítem acima, mas foi usada a biblioteca LZO ao invés da Zlib. A diferença entre os dois métodos de compressão é que a LZO foca em velocidade, comprometendo parte da taxa de compressão.

- Precache - Na máquina remota estava disponível uma instalação da mesma distribuição a ser transferida. Ao ser enviado o *hash* do disco virtual, a máquina remota encontrou todos os blocos localmente para remontar o disco, não necessitando requisitar blocos de dados da máquina de origem.

Cada experimento será realizado três vezes, e calculada a média.

A figura 5.2 mostra que, em uma rede veloz, o método Hash oferece apenas uma pequena vantagem com relação ao tempo de transferência. Isto ocorre porque o tempo de processamento necessário para pesquisar por blocos idênticos e recriar a estrutura do disco é semelhante ao tempo de simplesmente transferir todos os dados. Em contrapartida, ao utilizar parte da versão 9.1, 307MB deixaram de ser transferidos pela rede, o que é benéfico em uma rede compartilhada, onde se espera haver outros tráfegos simultâneos. O método Precache se mostrou mais eficiente que

[1] Como há diversos outros usuários, tanto no LNCC, quanto na UFRJ, compartilhando o mesmo *link*, não é possível determinar com precisão a real largura de banda disponível.

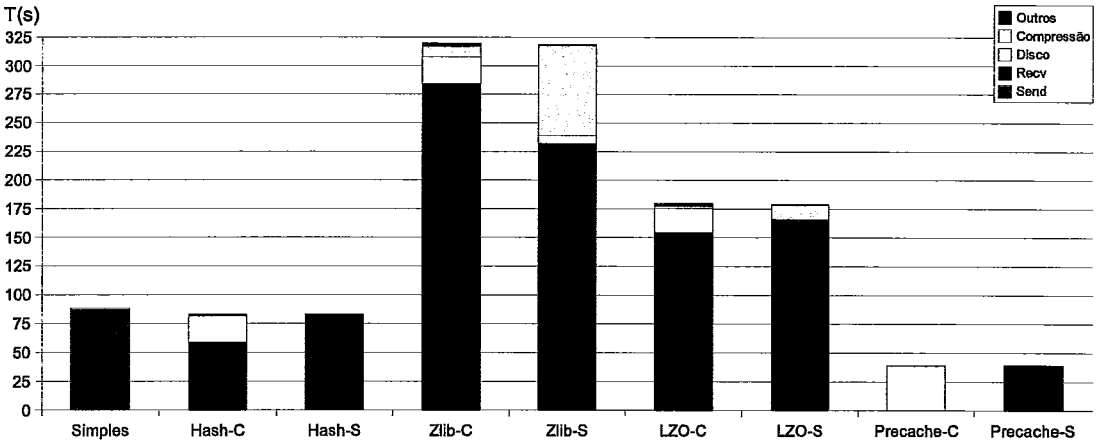


Figura 5.2: Simulação com 100Mbps de banda.

quaisquer outros, além de enviar apenas 5MB de dados, aproximadamente 0,5% do total dos dados a transferir.

O uso de compressão acarretou em maior sobrecarga de tempo, com relação às demais técnicas, porque a sobrecarga de processamento é mais alta que a de comunicação. A técnica Hash-LZO se sai melhor que a Hash-Zlib, porque a maior taxa de compressão obtida pela Zlib não compensa a diferença de tempo de processamento necessário à compressão.

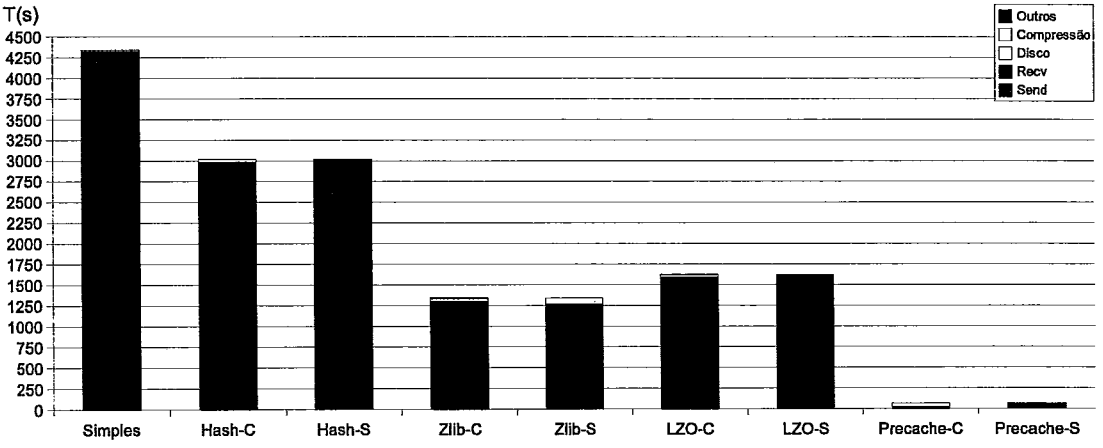


Figura 5.3: Simulação com 2Mbps de banda.

A figura 5.3, referente aos experimentos em uma banda disponível de 2Mbps, mostra a eficiência de métodos mais inteligentes na transferência de grandes quantidades de dados, em uma rede com banda restrita.

A técnica mais eficiente é, nos quesitos tempo e quantidade de dados transmitidos, a Precache, necessitando de pouco mais de um minuto para remontar remota-

mente um disco idêntico ao da máquina de origem. Enquanto que a menos eficiente é a Simples, já que transmite todo o conteúdo do disco pela rede.

Aplicando a técnica de *hash*, o tempo já é reduzido consideravelmente, em relação à transferência Simples, além de reduzir a quantidade de dados a serem transferidos na rede. Pela baixa velocidade da rede, a compressão da Zlib é mais eficiente que a LZO, pois o tempo de processamento não é mais tão crítico quanto no caso de uma rede veloz.

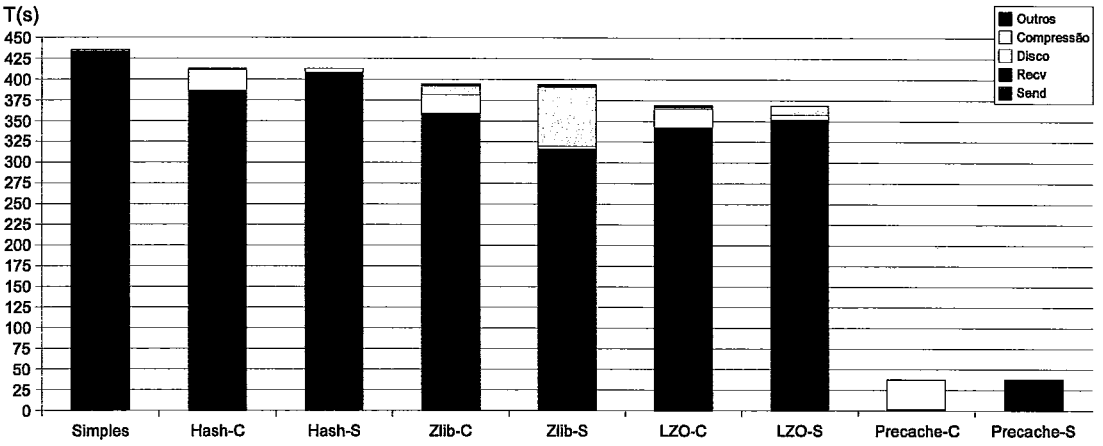


Figura 5.4: Teste entre o LNCC e o LCP.

O teste referente à figura 5.4, realizado entre o LNCC e o LCP, aponta um ponto de equilíbrio entre largura de banda e uso de CPU, que pode ser observado comparando o tempo de execução do método simples e do método utilizando *hashing*. Quando se adiciona o uso de compressão, o tempo diminui sutilmente, compensando, obviamente, com a redução da quantidade de dados a serem transferidos. A técnica precache, novamente, se mostrou a mais eficiente.

5.4.1 Versão iterativa

Neste experimento, avaliamos o *overhead* introduzido pelo cálculo *on-the-fly* do *hash* do disco virtual, utilizado na versão iterativa. A tabela 5.2 mostra o tempo necessário para calcular o *hash* de todos os blocos dos discos virtuais utilizados nos experimentos, bem como o tamanho em *megabytes* de cada disco. O tempo acrescido na transferência é de duas vezes o tempo de cálculo do *hash* mais o tempo necessário para transferir as modificações feitas no disco durante a primeira iteração.

Versão	Tamanho(MB)	Tempo(s)
9.0	706	15,87
9.1	910	20,48
10.0	990	21,7

Tabela 5.2: Cálculo do *hash* do disco virtual

5.5 Discussão dos Resultados

Com os *benchmarks* NAS, vimos a baixa sobrecarga do Xen, mostrando uma diferença de menos de 2%, comparada à execução sobre o Linux nativo.

Com a identificação de similaridades na estrutura física dos discos contendo instalações de sistemas Linux, verificamos a aplicabilidade da técnica de comparação de *hash* e vimos que uma parte considerável dos blocos de *releases* diferentes é mantida, para permitir a compatibilidade com versões anteriores dos sistemas.

A técnica de transferência dos discos virtuais, utilizando *hashing*, mostrou-se bastante eficiente. Em um ambiente de *grid*, os benefícios obtidos são a redução do tempo de transferência e redução do tráfego na rede, enquanto que em um *cluster* ou rede local, o ganho de desempenho é modesto, devido ao tempo da computação ser próximo ao de simplesmente transferir todo o disco através da rede. No entanto, a redução do tráfego na rede, devido ao uso das técnicas implementadas, acarretará em economia também de tempo, tanto para a aplicação sendo migrada, quanto para as demais que estiverem utilizando a rede.

Com o constante aumento de capacidade e barateamento dos discos rígidos disponíveis no mercado, é razoável afirmar que o método precache não está longe da realidade, e poderá ser de grande valia na migração de máquinas virtuais em ambientes de grades computacionais.

Capítulo 6

Trabalhos Relacionados

6.1 Máquinas virtuais e grids

Figueiredo et al.[22] propõem a utilização de máquinas virtuais em grids. Nesse trabalho, as máquinas virtuais funcionam como nós do *grid*, recebendo submissão de processos, enquanto que nesta dissertação, as máquinas virtuais são vistas como o processo do usuário sendo submetido. Também descrevem a possibilidade de migração e suas vantagens, mas não são dados detalhes ou uma implementação.

6.2 Migração

Sapuntzakis et al.[36] descrevem um sistema de cápsulas, que são um conjunto de *snapshots* de estados de uma máquina virtual, com o objetivo de realizar a migração apenas das modificações feitas. O sistema é implementado utilizando como base o VMWare ESX, e realiza a transferência de páginas sob demanda. O enfoque do trabalho é realizar a transferência de máquinas virtuais "pessoais", entre ambientes de trabalho, por exemplo. Para grades computacionais, a abordagem traz complicações, por manter dependência entre a máquina virtual e máquinas hospedeiras.

ZAP[32] é uma camada de virtualização sobre o sistema operacional (Linux) que provê a abstração de PODs. PODs são um conjunto de processos e dependências de arquivos e conexões de rede, e podem ser migrados como uma máquina virtual para outro sistema Linux. A migração se dá de forma eficiente, e com baixo overhead. A diferença entre a abordagem de migração de processos e de máquinas virtuais é a independência parcial de um, contra a total independência do outro com relação ao sistema da máquina hospedeira. Não é possível migrar um POD de uma máquina para outra que possua um *kernel* ou bibliotecas de sistema incompatíveis.

Xen oferece mecanismos para *live migration*[16] de máquinas virtuais, conseguindo realizar a migração de domínios Xen entre nós de um cluster com um baixo tempo de indisponibilidade. No entanto, o sistema assume que os sistemas de arquivos estão disponíveis em um servidor de discos, não realizando a migração do sistema de arquivos, apenas da memória.

Capítulo 7

Conclusões

Nesta dissertação, foi apresentada uma proposta de utilização de virtualização em ambientes de grades computacionais para a execução de aplicações científicas CPU *bound*.

Mostramos uma visão geral de grades computacionais e seu estado atual, apontando suas vantagens e algumas deficiências.

Apresentamos uma visão geral de máquinas virtuais: suas aplicações em âmbito geral; técnicas de implementação, casos em que são melhor aplicadas e suas desvantagens; além do estado-da-arte em virtualização: o Xen.

Mostramos os benefícios alcançados com a introdução da virtualização em ambientes de grades computacionais: a independência do *software* presente na máquina hospedeira e a liberdade do usuário em administrar o ambiente computacional em que executará sua aplicação; a maior segurança obtida com o isolamento das máquinas virtuais entre si, e entre máquinas virtuais e sistema operacional hospedeiro. Para demonstrar a viabilidade prática do uso da virtualização em *grids*, utilizamos os *benchmarks* do NAS e confirmamos a baixa sobrecarga de processamento advinda da virtualização.

Desenvolvemos e implementamos mecanismos para a migração de máquinas virtuais, que permitem o uso de técnicas de tolerância a falhas e balanceamento de carga, além de avaliarmos seu desempenho em diferentes cenários de *grid*. A utilização destes mecanismos se mostrou vantajosa, diminuindo significativamente o tráfego de dados na rede durante a migração das máquinas virtuais, além de reduzir consideravelmente o tempo da migração, quando o cenário é favorável às técnicas implementadas (tipicamente, os menos propícios à migração).

Demonstramos a presença de características em sistemas de arquivos e distribui-

ções Linux que suportam a aplicabilidade das técnicas desenvolvidas.

Como trabalhos futuros, sugerimos:

- Introdução do Xen e técnicas desenvolvidas em um *grid middleware*;
- Estudo da aplicação das técnicas em casos específicos de aplicações e avaliação de repetibilidade dos dados do usuário.

Bibliografia

[1] Bochs ia-32 emulator project. http://bochs.sourceforge.net.

[2] The linux traffic shaper. http://lwn.net/1998/1119/shaper.html.

[3] Lvm. http://sourceware.org/lvm2/.

[4] The new plex86 x86 virtual machine project. http://plex86.sourceforge.net.

[5] Ntfs - new technology file system. http://www.ntfs.com/.

[6] Us secure hash algorithm 1. http://tools.ietf.org/html/rfc3174.

[7] User-mode linux. http://user-mode-linux.sourceforge.net.

[8] Virtual private server. http://en.wikipedia.org/wiki/Virtual_private_server.

[9] Vmware. http://www.vmware.com.

[10] Inc. Advanced Micro Devices. Amd's virtualization solutions. http://enterprise.amd.com/us-en/Solutions/Consolidation/virtualization.aspx.

[11] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks. *The International Journal of Supercomputer Applications*, 5(3):63–73, Fall 1991.

[12] A. Barak and O. La'adan. The mosix multicomputer operating system for high performance cluster computing, 1998.

[13] Fabrice Bellard. Qemu. http://fabrice.bellard.free.fr/qemu/.

[14] Florian Buchholz. The structure of the reiser file system. http://homes.cerias.purdue.edu/ florian/reiser/reiserfs.php.

[15] Bryan Clark, Todd Deshane, Eli Dow, Stephen Evanchik, Matthew Finlayson, Jason Herne, and Jeanna Neefe Matthews. Xen and the art of repeated research. In *USENIX Annual Technical Conference, FREENIX Track*, pages 135–144, 2004.

[16] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 273–286, Boston, MA, May 2005.

[17] Intel Corporation. Intel virtualization technology. www.intel.com/technology/computing/vptech/.

[18] Microsoft Corporation. Fat32 file system specification. http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx.

[19] Microsoft Corporation. Microsoft ntfs technical reference. http://www.microsoft.com/technet/.

[20] L. Deutsch. Zlib compressed data format specification version 3, 1950.

[21] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, October 2003.

[22] R. Figueiredo, P. Dinda, and J. Fortes. A case for grid computing on virtual machines, 2003.

[23] I. Foster. Globus toolkit version 4: Software for service-oriented systems. *IFIP International Conference on Network and Parallel Computing*, Springer-Verlag LNCS 3779:2–13, 2005.

[24] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[25] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[26] Helena Handschuh, Lars R. Knudsen, and Matthew J. Robshaw. Analysis of SHA-1 in encryption mode. *Lecture Notes in Computer Science*, 2020:70–83, 2001.

[27] Network Working Group R. Housley. Request for comments: 3378, 2002.

[28] Eliane Araújo Gustavo Mendes Roberta Coelho Walfredo Cirne Lauro Beltrão Costa, Loreno Feitosa and Daniel Fireman. Mygrid: A complete solution for running bag-of-tasks applications. *22nd Brazilian Symposium on Computer Networks - III Special Tools Session*, May 2004.

[29] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[30] Athicha Muthitacharoen, Benjie Chen, and David Mazières. A low-bandwidth network file system. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 174–187, Chateau Lake Louise, Banff, Canada, October 2001.

[31] Markus F.X.J. Oberhumer. Lzo real-time data compression library. http://www.oberhumer.com/opensource/lzo/.

[32] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The design and implementation of zap: A system for migrating computing environments, 2002.

[33] Eduardo Pinheiro and Ricardo Bianchini. Nomad: A scalable operating system for clusters of uni and multiprocessors. In *Proceedings of the 1st IEEE International Workshop on Cluster Computing*, December 1999.

[34] The FreeBSD Documentation Project. Freebsd handbook. http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/index.html.

[35] Stephen Tweedie Rémy Card, Theodore Ts'o. Design and implementation of the second extended filesystem. In *First Dutch International Symposium on Linux ISBN 90-367-0385-9.*

[36] Constantine P: Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, and Mendel Rosenblum. Optimizing the migration of virtual computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.

[37] A. Whitaker, M. Shaw, and S. Gribble. Denali: Lightweight virtual machines for distributed and networked applications, 2002.