4

Virtualização: Princípios Básicos e Aplicações

Professor:
Alexandre Carissimi[1]
(asc@inf.ufrgs.br)

Resumo:

Virtualização é abstração que representa um recurso computacional qualquer, mas sua forma mais conhecida é através de máquinas virtuais. Uma máquina virtual oferece um ambiente completo, muito similar ao de uma máquina física real, tendo seu próprio sistema operacional, aplicativos e serviços de rede (Internet) de forma totalmente isolada e independente uma das outras. Devido a essa característica, a virtualização tem recebido uma atenção especial em infra-estruturas de TI para efetuar a consolidação de servidores, reduzir custos de gerenciamento e de energia, prover tolerância a falhas e melhorar a segurança. O objetivo deste trabalho é fornecer os conceitos básicos para compreender no quê consiste a virtualização, suas formas de implementação, compromissos e vantagens de suas aplicações típicas em uma infra-estrutura de TI e de processamento paralelo .

[1] Graduado em Engenharia Elétrica pela Universidade Federal do Rio Grande do Sul (1985), mestre em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (1989) e doutor em Informática - Institut National Polytechnique de Grenoble (1999). Atualmente é professor 40/DE da Universidade Federal do Rio Grande do Sul (UFRGS), lotado no Departamento de Informática Aplicada. Atua principalmente nos seguintes temas: redes de computadores, sistemas operacionais, sistemas distribuídos, ambientes de programação paralela e ambientes de programação.

4.1. Introdução

A virtualização e o uso de máquinas virtuais se tornaram um assunto destaque no mundo da tecnologia da informação (TI), apesar de não ser exatamente uma novidade. A introdução da linguagem de programação Java nos anos 90 trouxe consigo a noção de máquina virtual, mas sua origem é mais antiga. O termo **máquina virtual** foi introduzido na década de 60 como um conceito de sistemas operacionais para indicar uma abstração em software de um sistema computacional em hardware. Já nos anos 70 era comum que cada computador (*mainframe*), mesmo de um único fabricante, tivesse seu próprio sistema operacional causando problemas de portabilidade e de sistemas legados. Na época, a solução encontrada foi explorar a risca o conceito de máquina virtual, ou seja, uma camada de software que oferece um ambiente completo similar ao de uma máquina física que executa sobre um sistema computacional nativo. Uma máquina virtual contém seu próprio sistema operacional, bibliotecas e aplicativos e é totalmente independente e isolada das demais. Essa abordagem foi usada com sucesso pela IBM que, na linha de *mainframes* 370 e seus sucessores, oferecia uma máquina virtual portada para cada uma das plataformas de hardware sobre a qual as aplicações executavam. Dessa forma era possível executar, ou migrar, uma aplicação, de uma plataforma a outra, desde que houvesse uma versão de máquina virtual para a plataforma alvo.

Na década de 80, a medida que os computadores começaram a se tornar mais comuns e terem seu hardware uniformizado, a quantidade de sistemas operacionais convergiu para algumas poucas famílias (Unix, Macintosh e Microsoft), cada uma com um público-alvo e um conjunto de aplicativos. Nesse contexto, o emprego de máquinas virtuais perdeu importância. Entretanto, o aumento do poder computacional dos atuais processadores, a disseminação de sistemas distribuídos e a onipresença das redes de computadores causaram, por várias razões, o ressurgimento da virtualização.

Até agora, a virtualização foi apresentada como uma técnica que permite a execução de múltiplos sistemas operacionais e de suas aplicações em máquinas virtuais sobre uma única máquina física. Entretanto, o conceito de virtualização é mais amplo. Segundo a EMA (*Enterprise Management Association*), virtualização é a técnica que "mascara" as características físicas de um recurso computacional dos sistemas, aplicações ou usuários que os utilizam. Nesse contexto, encontramos a virtualização na implementação de *desktops* remotos, de discos virtuais, na computação em *cluster* e mesmo em dados como, por exemplo, através do uso de XML, SQL, JMS, entre outros. Ainda, a partir do momento que se define máquinas virtuais, surge, quase que imediatamente, a necessidade de conectá-las em rede. Todas as máquinas virtuais existentes provêm interfaces de redes virtuais idênticas as suas similares reais, isso é, com endereços MAC e podem ser interligadas a equipamentos de interconexão de rede reais, ou virtuais, como *switches* e roteadores. Outro ponto interessante de uma máquina virtual é que, por ela ser um ambiente isolado, o comprometimento de sua segurança não afeta as demais. Inclusive, é possível ter políticas de segurança diferentes para cada uma delas.

Com base no que foi mencionado até o momento, é possível imaginar que a virtualização oferece um potencial muito grande para criar e manter infra-estruturas de rede. O objetivo deste trabalho é fornecer os principais conceitos da virtualização e de suas formas de implementação. Atualmente existem várias ferramentas que oferecem suporte a virtualização, tanto soluções proprietárias, quanto em software livre. As mais

conhecidas são Xen. VMware, Microsoft VirtualPC 2007 e Virtualbox, e serão apresentadas como estudos de caso. Conclui-se com a discussão de algumas aplicações típicas da virtualização em infra-estruturas de TI e processamento paralelo.

4.2. Fundamentação teórica

Um computador é um sistema de computação relativamente complexo devido à variedade de componentes de hardware e de software que o constitui e às interações entre eles. Entretanto, essa complexidade nunca foi um empecilho para o crescimento e a evolução dos computadores em geral devido ao seu projeto de forma hierárquica, com diferentes níveis de abstração e com interfaces bem definidas entre esses níveis. O uso de níveis de abstração e interfaces, tanto para os componentes de software, como para os de hardware, permitiu que cada componente fosse visto como um subsistema independente oferecendo serviços para os demais. Os detalhes internos de implementação de cada um deles não precisam ser conhecidos: basta conhecer as interfaces e os serviços oferecidos.

O princípio básico é o de "dividir para conquistar", o que significa dividir o sistema computacional em várias camadas funcionais hierárquicas, cada uma com um nível de abstração apropriado e provendo serviços para a camada superior através de uma interface bem definida. Uma camada *n* não enxerga os detalhes internos da camada *n-1*, mas apenas as suas abstrações e interfaces. Por exemplo, é possível saber o quê um processador é capaz de executar analisando seu conjunto de instruções *assembly* sem conhecer como ele é internamente construído. Isso permite que um compilador gere código compatível para processadores de fabricantes diferentes, desde que eles ofereçam a mesma interface, isso é, o mesmo conjunto de instruções com os mesmos códigos de máquina. Da mesma forma, programadores C não precisam estar cientes de detalhes do hardware e do sistema operacional quando eles usam recursos padrões da linguagem.

Entretanto, essa organização cria uma dependência entre as camadas em função das interfaces disponibilizadas, ou seja, é obrigatório respeitar uma interface para usar os serviços daquela camada. Uma solução para eliminar essa dependência seria mapear uma interface para outra com a introdução de uma camada intermediária de adaptação. Esse mapeamento é a base da virtualização e foi introduzido através do conceito de isomorfismo [Popek e Goldberg, 1974]. O **isomorfismo** consiste em transformar o estado de um sistema *A* em um estado equivalente em sistema *B*. Para uma seqüência *s* de operações que modificam o estado E_i em um estado E_j no sistema *A*, existe uma seqüência *s'* que executam uma transformação equivalente do estado E'_i para o estado E'_j no sistema *B*. Apesar do isomorfismo também poder ser usado para explicar o que é abstração, [Smith e Nair 2005] fazem uma diferenciação interessante entre abstração e virtualização: enquanto uma abstração esconde detalhes internos de um componente, a virtualização os apresenta ao exterior de uma forma diferente da real.

Um computador é composto por duas grandes camadas, a de hardware e a de software, que por sua vez são divididas em subcamadas. Para melhor compreender o princípio de funcionamento da virtualização, e os tipos de máquinas virtuais existentes, é preciso conhecer aspectos básicos dessas duas camadas. Esses aspectos dizem respeito a alguns conceitos básicos de arquitetura de computadores e de sistemas operacionais, os quais serão apresentados a seguir.

2. 1 Aspectos de arquitetura de computadores

Por arquitetura de computadores entende-se a descrição lógica e funcional dos componentes que formam o hardware de um sistema computacional e suas interações,

mas sem entrar em detalhes de como cada um desses componentes é implementado internamente. Segundo [Hennessy e Patterson 2007], a área de arquitetura de computadores engloba pelo menos três grandes categorias:

- Conjunto de instruções de máquina *(Instruction Set Architecture* - ISA): é a abstração do processador através de seu conjunto de instruções de máquina (*assembly*). Inclui, além das instruções, os modos de endereçamento possíveis e os registradores de máquina existentes.
- Projeto do sistema: envolve os componentes externos ao processador como barramentos, memória, controladores, arbitramento, subsistema de E/S e suas interconexões. Trata ainda dos mecanismos de suporte necessários a multiprocessadores, por exemplo.
- Microarquitetura: descrição de como são constituídas as unidades internas de um processador e como elas são interligadas para implementar o conjunto de instruções. Diz respeito a como um processador é organizado internamente.

Cada uma dessas categorias possui um nível de abstração que fornece um conjunto de serviços e uma interface. Apenas as duas primeiras categorias, conjunto de instruções e projeto de sistema, são importantes no contexto deste capítulo. A figura 4.1, fornece uma visão esquemática das diferentes camadas de abstração encontradas em um sistema computacional incluindo o sistema operacional e seus aplicativos.

O conjunto de instruções (ou ISA) é a interface limítrofe entre o nível de abstração de hardware e o de software. Essa interface é composta por todos os códigos de máquina aceitos pelo processador e que correspondem, cada um, a uma instrução. Tipicamente, um processador possui pelo menos dois modos de operação, não-privilegiado e privilegiado[1]. A interface ISA é dividida em dois subconjuntos para refletir cada um desses modos de operação. O primeiro é formado por todas as instruções de máquina que podem ser diretamente executadas por programas de usuário e, por isso, constitui o que denomina de instruções de usuário ou *user* ISA. Os programas de usuário executam em modo não-privilegiado. O segundo subconjunto, formado pelas instruções de sistema (*system* ISA), é constituído por aquelas instruções de máquina capazes de configurar o comportamento do próprio processador e de acessar componentes de hardware diretamente, como as instruções de E/S. Essas instruções são acessíveis unicamente em modo privilegiado e, na prática, são executadas exclusivamente pelo núcleo do sistema operacional. A *user* ISA e a *system* ISA estão assinaladas na figura 4.1 como interfaces 1 e 2, respectivamente.

Ao iniciar um sistema computacional, os primeiros passos realizados antes de se passar o controle ao sistema operacional estão relacionados com a configuração do processador e de seu hardware. Isso é feito através da inicialização de estruturas internas e de registradores de controle do processador como, por exemplo, vetor de interrupção e a unidade de gerência de memória (*Memory Management Unit* – MMU), e das controladoras dos dispositivos de hardware. Tipicamente, cada controladora possui registradores internos que permitem sua configuração, leitura e escrita de dados e a consulta de seu estado de operação. Esses registradores são vistos pelo processador como uma série de endereços especiais definidos no momento do projeto e constituem uma interface entre o processador e os dispositivos de hardware. Na figura 4.1, essas interfaces são assinaladas como 3, 4 e 5. As instruções de máquina que permitem configurar o processador e acessar endereços especiais de E/S pertencem ao conjunto de

[1] Também denominados, respectivamente, de modo usuário e modo sistema ou ainda modo não-protegido e modo protegido.

instruções de sistema (*system* ISA). Note que, após a inicialização, o sistema operacional tem acesso ao hardware, ao sistema de memória, ao processador e aos dispositivos de E/S através dessas instruções.

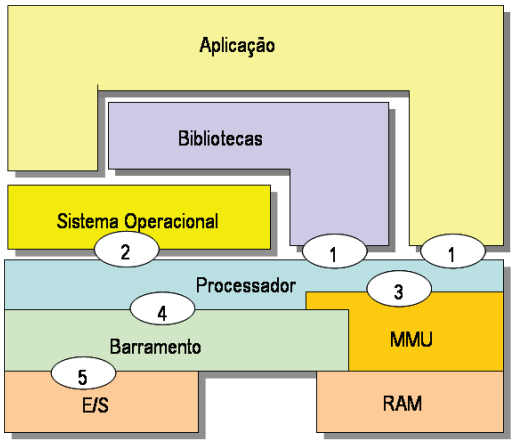

Figura 4.1 – Arquitetura de computadores como um conjunto de camadas de abstração (adaptada de [Smith e Nair 2005])

4.2.2 Aspectos de sistemas operacionais

Qualquer pessoa que atualmente use um computador sabe que existe algo denominado de sistema operacional que, de alguma forma, controla os diversos dispositivos que o compõe. A definição clássica para sistema operacional, encontrada em vários livros, é a de uma camada de software inserida entre o hardware e as aplicações que executam tarefas para os usuários e cujo objetivo é tornar a utilização do computador, ao mesmo tempo, mais eficiente e conveniente [Silberschatz, 2001].

A utilização mais eficiente busca um maior retorno no investimento feito no hardware. Maior eficiência significa mais trabalho obtido pelo mesmo hardware. Isso é obtido através da distribuição de seus recursos (espaço em memória principal, processador, espaço em disco, etc) entre diferentes programas. Cada programa tem a ilusão de estar executando sozinho no computador quando na realidade ele está compartilhando com os demais. Uma utilização mais conveniente do computador é obtida, escondendo-se do usuário detalhes de hardware, em especial, dos periféricos de entrada e saída. Tipicamente, isso é feito através da criação de recursos de mais alto nível oferecido através de interfaces gráficas. Por exemplo, os usuários usam espaço em disco através do conceito de arquivos. Arquivos não existem no hardware. Eles formam um recurso criado a partir do que o hardware oferece. Isso é um exemplo de virtualização de recursos.

O conceito fundamental em sistemas operacionais é o de processo. Um processo é uma abstração que representa um programa em execução. Um processo é representado por um espaço de endereçamento lógico composto por regiões de texto e de dados, uma pilha e um *heap* (figura 4.2). A região de texto contém o código a ser executado. A região de dados mantém todas as variáveis globais, inicializadas ou não. A pilha serve para armazenar o endereço de retorno de uma chamada de função, para passagem de parâmetros, além de ser também a área de memória onde são criadas as variáveis locais. Por fim, a região de *heap*, que serve para a alocação dinâmica de porções de memória.

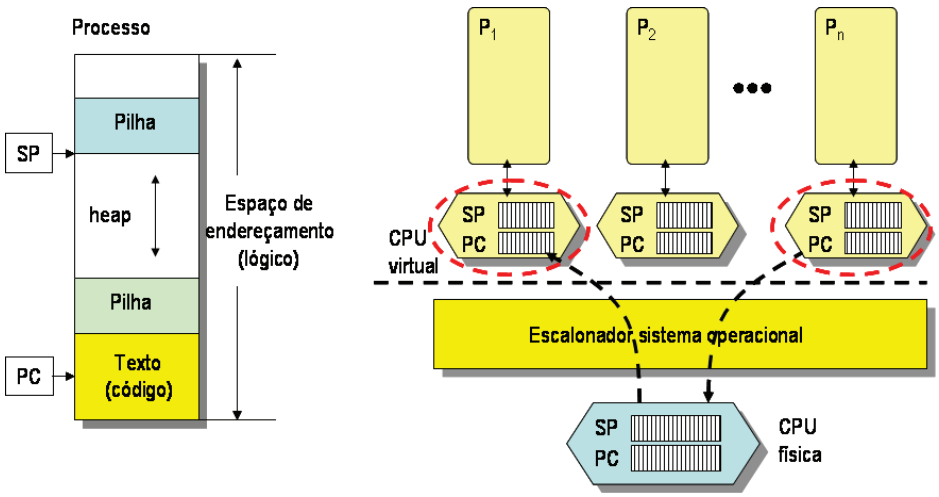

Figura 4.2 – A abstração de processo em um sistema operacional

Cada processo é um ambiente de execução isolado dos demais processos que executa sobre um processador lógico, isto é, um processador virtual, vinculado a si no momento da criação do processo[2]. Cabe ao núcleo do sistema operacional, através de seu escalonador, alternar os diferentes processadores lógicos (virtuais) sobre um processador físico. A ilusão de paralelismo é criada pelo chaveamento rápido entre os processos.

De uma forma simplificada, a execução de um processo pode ser acompanhada pela abstração de dois registradores lógicos (virtuais): o contador de programa (*Program Counter* - PC) e o apontador de pilha (*Stack Pointer* – SP). O contador de programa fornece, em um dado instante de tempo, um endereço da região de texto onde se encontra a instrução a ser executada. O apontador de pilha informa a região onde devem ser armazenados o endereço de retorno de uma chamada de função, seus parâmetros e suas variáveis locais. É através do apontador de pilha que se lê e escreve nos parâmetros de função e nas variáveis locais a função. Cabe ao sistema operacional, através de seu escalonador e do *dispatcher*, mapear os registradores lógicos PC e SP de um processo para os registradores físicos PC e SP – e únicos – do processador durante o chaveamento de contexto[3]. A alternância entre os diversos PC e SP lógicos, dos diferentes processos, nos registradores PC e SP físicos fornece a ilusão de que vários processos estão executando simultaneamente (figura 4.2). Portanto, um processo nada mais é que um tipo de máquina virtual que executa um único programa.

Outro princípio importante em sistemas operacionais é a sua estruturação em camadas hierárquicas, com seus diferentes níveis de abstrações e interfaces, de forma similar ao que ocorre em nível de hardware. Inicialmente, na seção anterior, foi visto que um processo de usuário só pode empregar instruções não-privilegiadas (*user* ISA), entretanto, em certas situações, como para realizar E/S, os processos de usuário

[2] Na realidade, os sistemas operacionais atuais possuem duas abstrações para unidade de execução: processos e *threads*. Porém, continua válida a noção de um processador virtual por unidade de execução. Um estudo desses conceitos extrapola o escopo deste trabalho. O leitor pode obter detalhes em [Silberschatz, 2001][Oliveira, Carissimi e Toscani, 2004].

[3] O chaveamento de contexto é mais complexo do que simplesmente agir sobre os registradores SP e PC, pois envolve salvamento de estado de todos os registradores físicos do processador e a atualização de várias tabelas internas, mas para esta discussão, essa simplificação é suficiente.

precisam executar instruções do modo privilegiado. Para permitir isso, sem violar o princípio dos dois modos de operação, o sistema operacional disponibiliza aos programas de usuário um conjunto de chamadas de sistema (*system calls*). As chamadas de sistema possibilitam que os programas de usuários acessem de forma indireta e controlada, através do sistema operacional, os recursos de hardware. Ao fazer isso, cabe ao sistema operacional realizar todos os procedimentos necessários para garantir que o processo de usuário não comprometa o correto funcionamento do sistema. As chamadas de sistema constituem, portanto, uma interface entre o processo de usuário e o sistema operacional.

Como mencionado anteriormente, um dos objetivos de um sistema operacional é tornar mais conveniente o uso de um sistema computacional. Dentro desse contexto surgem as bibliotecas de funções, as quais disponibilizam uma série de funcionalidades para simplificar a construção de programas. Por exemplo, a biblioteca de *sockets* provê os mecanismos para o desenvolvimento de aplicações em rede ou, ainda, a biblioteca *gtk* que facilita a construção de interfaces gráficas. Cada função de biblioteca realiza um determinado procedimento e é identificada por um nome, parâmetros e valor de retorno. O conjunto de funções de uma biblioteca constitui o que é denominado de interface aplicativa ou apenas API, do inglês, *Application Programming Interface*. Algumas APIs de bibliotecas apenas encapsulam chamadas de sistemas para apresentá-las de uma forma mais amigável ao programador, enquanto outras podem ser mais complexas e envolver várias chamadas de sistema para realizar uma determinada funcionalidade.

Uma API é definida para ser usada em conjunto com uma linguagem de programação de alto nível. Assim, por exemplo, um programador C utiliza em seu código fonte chamadas de funções de uma biblioteca com sintaxe e semântica do C. A própria biblioteca é um programa escrito em uma linguagem de alto nível que, posteriormente, é compilada e gera um código objeto com a implementação de cada função que disponibiliza. O código objeto de uma biblioteca, junto com o código objeto resultante da compilação de um programa fonte, forma o código executável.

Para que os programas de usuário e as bibliotecas possam executar sobre um dado sistema operacional e processador (plataforma) é preciso respeitar as chamadas de sistema disponibilizadas e ter um código binário compatível com o do processador. Isso é obtido através da compilação do programa e das bibliotecas para essa plataforma. O produto final é um código executável que contém os equivalentes binários das instruções de usuário (*user* ISA) e das chamadas de sistema (*system calls*). A junção das versões binárias dessas duas interfaces é denominada de interface binária da aplicação, ou ABI (*Application Binary Interface*). Na prática, o que um programa em execução "enxerga" de uma plataforma é apenas a sua ABI (figura 4.3) e é isso que será a base para a construção de máquinas virtuais. Vale ressaltar que todas as funções de uma biblioteca são mapeadas ou para chamadas de sistema ou para instruções não privilegiadas.

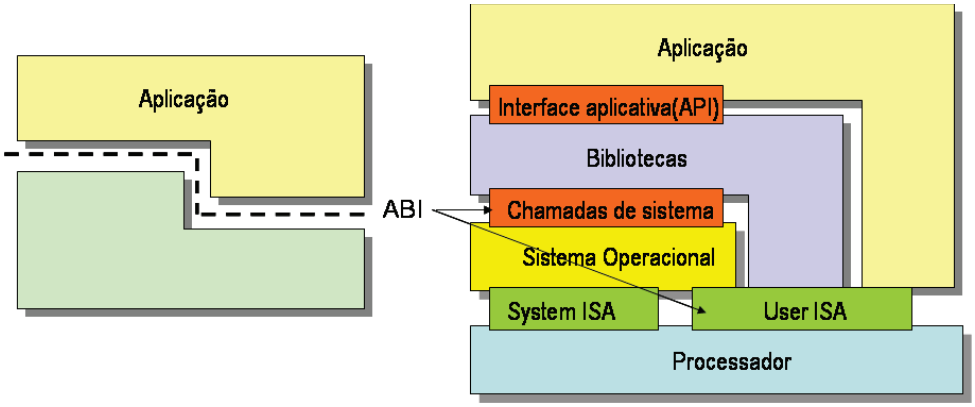

Figura 4.3 – Interfaces genéricas de um sistema de computação

4.2.3 O sistema operacional como uma máquina virtual

Um sistema operacional nada mais é que um software que executa sobre um determinado hardware com o objetivo de controlar seus recursos e oferecer um ambiente de execução para os programas aplicativos. Esse ambiente, definido no momento da criação do processo, é composto por um espaço de endereçamento contendo as instruções do programa de usuário a serem executadas e por uma versão lógica (virtual) dos registradores do processador real. Quando um processo é posto em execução pelo escalonador do sistema operacional, os valores dos registradores lógicos são atribuídos aos registradores reais e assim o programa é executado. Isso equivale a imaginar que cada vez que um processo é criado, o sistema operacional atribui a ele um processador virtual.

Além de virtualizar o processador, o sistema operacional oferece para os processos de usuário uma abstração dos recursos de hardware através do subsistema E/S e do sistema de arquivos. Em um sistema Unix, por exemplo, o subsistema de E/S classifica os dispositivos em orientados a bloco, orientados a caractere e rede. O acesso a cada dispositivo é feito através de chamadas de sistema como *put, get, ioctl* etc. O sistema de arquivos, através de sua noção de diretórios e arquivos, é como um usuário enxerga todos os dispositivos de armazenamento de dados (discos, cdrom, *pendrive*, por exemplo). O acesso ao sistema de arquivos é feito através de comandos e chamadas de sistema como *cd, ls, mv*, *open, read, write, close*, entre tantos outros. Portanto, o sistema operacional oferece para os processos de usuário um ambiente de execução que é basicamente uma máquina virtual completa (processador e dispositivos de E/S) cujo acesso é feito através das chamadas de sistema (a criação de processos também é uma chamada de sistema).

Na verdade, a máquina virtual disponibilizada para um processo de usuário pelo núcleo do sistema operacional é composta de duas partes. A primeira é justamente uma versão virtual da máquina real que o núcleo oferece através de chamadas de sistema e das suas abstrações de dispositivos de E/S e do sistema de arquivos. A segunda é o processador físico (real) visto pelas instruções de máquina que o processo executa (*user* ISA). Essas duas partes, como visto na seção anterior, compõem a interface binária de aplicação (ABI). Analisando sob esse ponto de vista, é possível explicar o por quê, por exemplo, de um aplicativo Windows não executar sobre um sistema operacional GNU/Linux: a máquina virtual oferecida não é a mesma que o aplicativo espera. Além disso, há o problema de compatibilidade de código binário devido as diferentes ISA. Por exemplo, um programa compilado para um Intel *x*86 compatível não tem como executar

sobre um *powerpc*. Note que tanto o núcleo do sistema operacional como o programa aplicativo contém instruções binárias específicas a um processador.

4.2.4 Virtualização

Em sua essência, a virtualização consiste em estender ou substituir um recurso, ou uma interface, existente por um outro de modo a imitar um comportamento. Isso é feito através de uma camada de software responsável por transformar ações de um sistema *A* em ações equivalentes em um sistema *B* (isomorfismo). Dependendo de como e onde essa transformação é feita, é possível classificar os softwares de virtualização em três grandes categorias [Rosemblum, 2004]:

- Nível de hardware: é aquela em que a camada de virtualização é posta diretamente sobre a máquina física e a apresenta às camadas superiores como um hardware abstrato similar ao original. Corresponde à definição original de máquina virtual dos anos 60.
- Nível de sistema operacional: é um mecanismo que permite a criação de partições lógicas em uma plataforma de maneira que cada partição seja vista como uma máquina isolada, mas que compartilham o mesmo sistema operacional. Nesse caso, a camada de virtualização se insere entre o sistema operacional e as aplicações. São exemplos desse tipo de abordagem o FreeBSD Jails, Linux Vserver, OpenVZ e as zonas do Solaris.
- Nível de linguagens de programação: a camada de virtualização é um programa de aplicação do sistema operacional. O objetivo é definir uma máquina abstrata sobre a qual executa uma aplicação desenvolvida em uma linguagem de programação de alto nível específica. Essa é a abordagem da máquina virtual Java e da *Microsoft Commom Language Infrastructure*, base do .Net.

Embora cada um desses tipos de virtualização tenha um objetivo diferente, eles possuem um conjunto comum de características desejáveis. Primeira, oferecer compatibilidade de software. Isso significa que todo software desenvolvido para uma máquina virtual deve executar nela independente de onde ela esteja sendo usada. Para isso acontecer, a máquina virtual deve mascarar as diferenças de software e de hardware existentes. Esse requisito traduz a filosofia Java de "*write once, run anywhere*". Segunda, isolamento, um software em execução em uma máquina virtual não deve ver, afetar ou ser afetado por outro software em execução em outra máquina virtual. Terceira, encapsulamento, permitir a qualquer momento a captura do estado completo do ambiente virtual parando a sua execução (*suspend*). Isso de ser feito de forma a possibilitar que, posteriormente, a execução seja retomada a partir desse estado (*resume*). Por fim, a camada de virtualização deve ser projetada e implementada de forma a não impactar em demasia o desempenho das aplicações que executam sobre ela. Isso representa uma relação custo-benefício que diz respeito a como as máquinas virtuais são implementadas. Esse assunto será desenvolvido nas próximas seções.

4.3. Máquinas virtuais

Para melhor compreender o conceito de máquina virtual é importante notar que existem duas perspectivas diferentes a serem consideradas: a de um processo e a do núcleo do sistema operacional. Sob o ponto de vista de um processo de usuário, uma *máquina* consiste em um espaço de endereçamento lógico onde as instruções e dados do programa residem. Os dispositivos de E/S existentes, e o tipo de acesso que se pode fazer a eles, são aqueles definidos pelo núcleo do sistema operacional e disponibilizados

através das chamadas de sistema. A interface entre a aplicação e a máquina virtual é feita através do conjunto de instruções não-privilegiadas (*user* ISA) e pelas chamadas de sistema (*system calls*). Com uma máquina virtual de processo, uma aplicação de usuário tem a ilusão de estar executando sozinha na máquina física.

Já na perspectiva do núcleo do sistema operacional, uma máquina virtual é um ambiente completo que oferece suporte de execução para várias aplicações (processos). Cabe ao núcleo do sistema operacional gerenciar a utilização dos recursos de hardware como memória, acesso a dispositivos de E/S e mesmo o processador entre os vários processos. Além disso, como visto anteriormente, o núcleo do sistema operacional ainda oferece uma visão abstrata dos recursos de hardware como, por exemplo, os meios de armazenamento através do sistema de arquivos. A interface entre o núcleo do sistema operacional e o hardware físico é feita através do conjunto de instruções (ISA) do processador. Diferentemente de um processo, que existe apenas enquanto está em execução, o núcleo do sistema operacional deve existir enquanto o sistema computacional estiver ativo. Essa constatação leva a afirmação de que uma máquina virtual de processo tem uma existência temporária enquanto que a máquina virtual de sistema é perene.

Cada uma dessas perspectivas dá origem a um tipo diferente de máquina virtual: as máquinas virtuais de processo e de sistema, que veremos a seguir. Antes, porém, alguns detalhes quanto à terminologia. O processo ou sistema operacional que executa sobre uma máquina virtual é denominado de **hóspede** ou **convidado**, enquanto que a plataforma subjacente onde a máquina virtual executa é denominada de **hospedeiro** ou **sistema nativo**. A camada de virtualização que implementa a máquina virtual é genericamente denominada de ***runtime*** (ou **executivo**) para as máquinas virtuais de processo, e de **monitor de máquina virtual** (*Virtual Machine Monitor* – VMM), ou **hipervisor** (*hypervisor*), para as máquinas virtuais de sistema.

4.3.1 Máquina virtual de processo

Uma máquina virtual de processo é aquela que fornece um ambiente de execução para uma única aplicação de usuário através de uma ABI virtual (chamadas de sistema e *user* ISA). Um ponto importante a destacar é que um processo é uma entidade efêmera, ou seja, ele existe apenas quando o programa está em execução. Portanto, uma máquina virtual de processo é criada sob demanda e só existe enquanto o processo estiver executando. Na figura 4.4, uma aplicação executa sobre um programa que implementa a máquina virtual de processo (*runtime* ou executivo). A máquina virtual utiliza as funcionalidades providas pelo sistema operacional – como chamadas de sistema e funções de biblioteca – e pelo próprio processador através de instruções não-privilegiadas. A aplicação emprega apenas a interface (ABI) provida pelo ambiente (máquina) virtual.

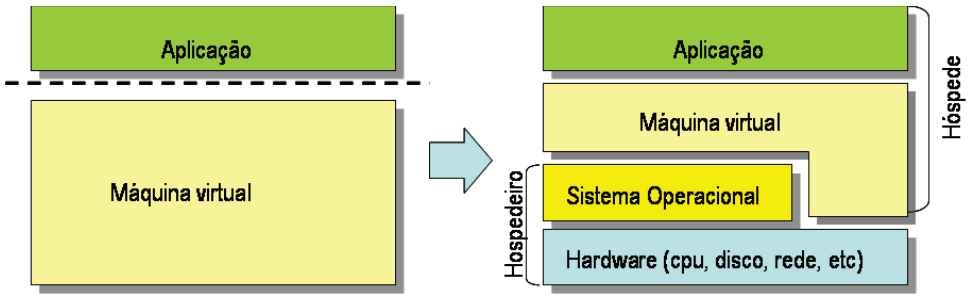

Figura 4.4 – Máquina virtual de processo

Um desafio importante na concepção de máquinas virtuais de processo é quando a aplicação hóspede possui um código binário para um processador diferente daquele sobre o qual a máquina virtual executa, o hospedeiro. Por exemplo, uma aplicação compilada para *Intel* 32 que se deseja executar sobre um processador *sparc*. Existem várias maneiras de resolver esse problema, a mais direta é através da interpretação. Um interpretador é um programa que realiza um ciclo de busca de instrução, decodificação e emulação dessa instrução para o sistema hospedeiro. Um exemplo típico dessa abordagem é a linguagem de programação Java e de sua máquina virtual (*Java Virtual Machine* -JVM). A JVM é uma máquina virtual de processo que possui um código binário específico, os *bytecodes*, para os quais uma aplicação Java é compilada. Ao se lançar uma aplicação Java, na verdade, se está executando a máquina virtual Java passando como parâmetro um arquivo binário *bytecode*. A máquina virtual Java então interpreta, *bytecode* a *bytecode*, transformado-os em ações e instruções equivalentes da máquina real subjacente (hospedeiro). A principal vantagem desse método é a portabilidade, mas o processo de interpretação pode ser bastante oneroso em termos de desempenho.

A tradução binária dinâmica surge como uma solução para melhorar o desempenho da interpretação feita nas máquinas virtuais de processo. Nesse caso, blocos de instruções da aplicação hóspede são traduzidos em blocos de instruções do sistema hospedeiro, o que já pode levar a alguma otimização em relação a se fazer instrução por instrução. Além disso, os blocos traduzidos podem ser armazenados em uma *cache* interna e posteriormente reaproveitados amortizando assim o custo da tradução. Essa técnica é denominada de **compilação *Just-In-Time*** (JIT).

3.2 Máquina virtual de sistema

Uma abordagem alternativa para a camada de virtualização é a definição de uma máquina virtual de sistema. Aqui a máquina virtual oferece um ambiente de execução completo onde podem coexistir um sistema operacional e vários processos, possivelmente de diferentes usuários. Dessa forma, uma única plataforma de hardware pode executar múltiplos sistemas operacionais hóspedes, um em cada máquina virtual, simultaneamente. É esse tipo de abordagem que permite a consolidação de servidores que será apresentada na figura 4.6.

Existem duas formas básicas de implementação de máquinas virtuais de sistema ou hipervisores [Goldberg, 1973] [IBM, 2008]. Os hipervisores tipo I, ou nativos, são aqueles que executam diretamente sobre o hardware de uma máquina real e as máquinas virtuais são postas sobre ele (figura 4.5a). A função básica de um hipervisor nativo é compartilhar os recursos de hardware (processador, memória, meios de armazenamento e dispositivos de E/S) entre as diferentes máquinas virtuais de forma que cada uma delas tenha a ilusão de que esses recursos são privativos a ela. Esse tipo de hipervisor

corresponde ao originalmente implementado nos sistemas IBM no início da década de 70. Atualmente, o Xen e o VMware ESX Server são exemplos de hipervisores que adotam essa abordagem.

Os hipervisores tipo II, ou hóspedes, são caracterizados por executar sobre um sistema operacional nativo como se fossem um processo deste (figura 4.5b). Nesse caso, o que o hipervisor oferece é uma camada de virtualização composta por um sistema operacional hóspede, possivelmente diferente do sistema operacional nativo, e por um hardware virtual criado sobre os recursos de hardware oferecidos pelo sistema operacional nativo. São exemplos dessa abordagem o Vmware Player, VirtualBox e o MS VirtualPC 2007.

Um outro aspecto a ser considerado nos hipervisores é em relação ao quê é realmente virtualizado. Uma possibilidade é fornecer uma ABI (chamadas de sistema e *user* ISA) completamente diferente daquela do sistema nativo (figura 4.6a). Nesse caso, o hipervisor hóspede precisa emular todas as instruções executadas o que representa um custo no desempenho. Entretanto, essa estratégia permite que um sistema operacional – e suas aplicações – executem em plataformas diferentes daquelas para os quais ele foi concebido. Um exemplo dessa abordagem é o *VMware Fusion* que oferece um ambiente Windows para o sistema operacional MacOS X.

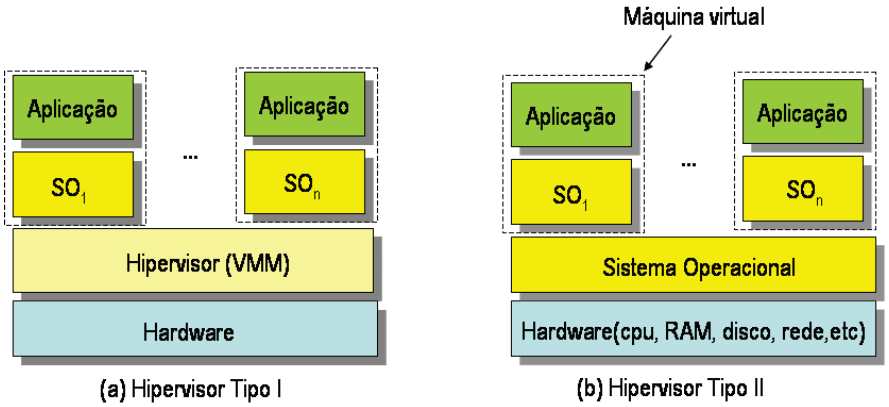

Figura 4.5 – Máquina virtual de processo: hipervisores tipos I e II

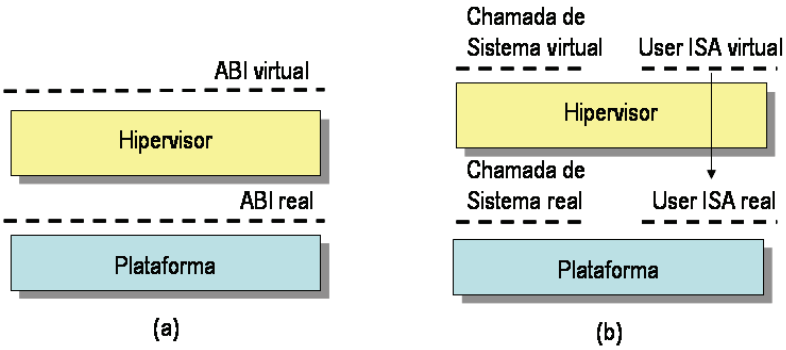

Figura 4.6 – Diferentes visões da virtualização

A segunda possibilidade considera que a máquina virtual (hipervisor) forneça um conjunto de instruções não-privilegiadas (*user* ISA) idêntico ao do sistema nativo sobre o qual executa (figura 4.6b). Nesse caso, não é necessário emular um conjunto de instruções a partir de outro. A virtualização, nesse caso, se reduz às chamadas de sistema e os recursos de hardware (discos, memória, etc) que compõem a máquina virtual. Esse é o caso, por exemplo, do VMwarePlayer, do VirtualBox e VirtualPC quando executam sobre um sistema operacional Windows e em processadores Intel IA-32 e oferecem à aplicação final um sistema operacional GNU/Linux.

4.4. Virtualização total e paravirtualização

A implementação das máquinas virtuais não é tão simples como possa aparentar em um primeiro momento. Além da preocupação direta com desempenho, existe o problema de como os recursos físicos da máquina são compartilhados entre o sistema nativo e o sistema hóspede sem que um interfira em outro, a começar pelo próprio processador. O problema fundamental consiste no que fazer quando o sistema hóspede executa instruções privilegiadas (*System* ISA), já que essas, por uma questão de proteção do sistema, são exclusivas ao sistema nativo (hospedeiro). A ação a ser tomada, depende de recursos oferecidos pela arquitetura do processador (*hardware*).

Em 1974, [Popek e Goldberg, 1974], classificaram o conjunto de instruções de um processador em três grupos: (1) privilegiadas, que se executadas por um programa em modo usuário causam exceções (*trap*); (2) sensíveis de controle, que permitem a alteração de recursos do sistema; e (3) sensíveis comportamentais, cujo resultado ou comportamento dependem da configuração de recursos como, por exemplo, conteúdo de registradores internos ou modos de execução do processador. Com base nessa classificação, Popek e Goldberg, propuseram o seguinte teorema: "*um monitor de uma máquina virtual pode ser implementado de forma adequada sempre que as instruções sensíveis de controle e sensíveis comportamentais forem um subconjunto das instruções privilegiadas*". Na prática, isso se traduz no fato que qualquer instrução que possa afetar o comportamento do sistema deve ser monitorada e tratada adequadamente.

Para melhor compreender esse problema que envolve o conjunto de instruções e a virtualização é interessante analisar a arquitetura x86. A arquitetura x86 provê quatro modos de operação para o processador, identificados de 0 a 3, denominados de anéis de proteção (*rings*) ou CPL (*Current Privilege Level*). Nos sistemas operacionais convencionais (Microsoft Windows e UNIXes), para esse tipo de arquitetura, apenas dois modos são usados. O *ring* 0, que detém os maiores privilégios de execução, é usado pelo sistema operacional, e o *ring* 3, de menor privilégio é empregado pelos processos de usuário. Se um processo de usuário tentar executar uma instrução privilegiada ocorrerá uma exceção (*trap*) que deverá ser tratada adequadamente. Entretanto, a arquitetura x86, em especial o Pentium, possui dezessete instruções sensíveis que são não privilegiadas [Robin, 2000], ou seja, programas em modo usuário podem afetar o funcionamento do processador sem gerar *traps*.

Entretanto, a virtualização é possível mesmo em processadores que não seguem a restrição enunciada no teorema de Popek e Goldberg, mas ao custo de um pior desempenho. A virtualização nessas arquiteturas é feita tratando as instruções sensíveis de acordo com duas estratégias: virtualização total e paravirtualização.

A **virtualização total** consiste em prover uma réplica (virtual) do hardware subjacente de tal forma que o sistema operacional e as aplicações possam executar como se estivessem diretamente sobre o hardware original (figura 4.7). A grande vantagem é

que o sistema operacional hóspede não precisa ser modificado para executar sobre o monitor de máquina virtual (VMM) ou hipervisor.

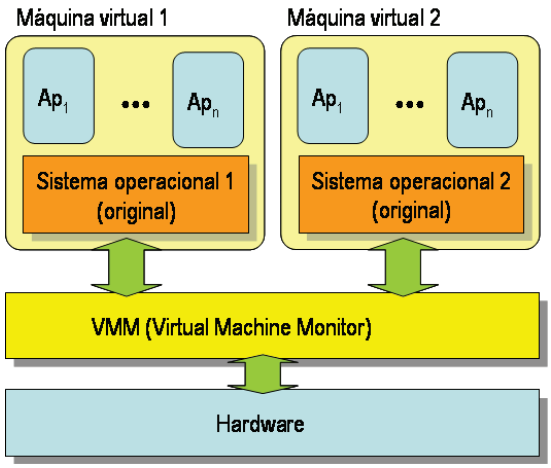

Figura 4.7 – Virtualização total

No entanto, a virtualização total tem alguns inconvenientes. Primeiro, por não ser modificado, todas instruções executadas pelo sistema hóspede devem ser testadas na máquina virtual para saber se elas são sensíveis ou não, o que representa um custo de processamento. As instruções sensíveis devem ser interceptadas e emuladas no hospedeiro para evitar que a máquina virtual altere o comportamento do sistema nativo. Essa interceptação e emulação podem ser onerosas se o processador nativo não possuir suporte em hardware para virtualização. Atento a isso, os fabricantes de processadores, Intel e AMD, desenvolveram extensões para arquitetura x86 para prover mecanismos que facilitassem essa tarefa. A Intel apresenta suas extensões para as arquiteturas x86 de 32 e 64 bits sob o nome IVT (*Intel Virtualization Technology*), ao passo que a AMD oferece esse suporte apenas para suas arquiteturas de 64 bits. A extensão da AMD é denominada de AMD-V, AMD-*Virtualization*. As soluções da Intel e da AMD foram desenvolvidas independentemente uma da outra e, embora sirvam para o mesmo propósito, são incompatíveis.

O segundo inconveniente da virtualização total é a dificuldade em se implementar uma máquina virtual que imite o comportamento exato de cada tipo de dispositivo, dada a diversidade de dispositivos existentes que compõem um computador. A solução consiste em prover a máquina virtual com suporte a um conjunto genérico de dispositivos. Tipicamente, cada máquina virtual possui um teclado e mouse do tipo PS/2, unidades de disquete, controladores IDE, cdrom ATAPI, portas seriais e paralelas, suporte USB, uma placa gráfica padrão e as placas de redes mais comuns em ambientes PC. Sendo assim, pode-se ter uma subutilização de um recurso de hardware real.

Por fim, a implementaçao de uma máquina virtual de processo deve contornar alguns problemas técnicos relativos a implementação da gerência de memória. Por exemplo, o Linux e o Windows implementam memória virtual através de paginação. Há toda uma gerência de alocação, liberação e controle de acesso às páginas que devem ser respeitadas. A questão que se apresenta é: deve a máquina virtual pré-alocar uma

quantidade de páginas do sistema nativo e emular sobre elas um espaço de endereçamento físico para o sistema hóspede ou pode-se usar diretamente a gerência do sistema nativo (hospedeiro)? Grosso modo, é necessário "converter" o espaço de endereçamento do sistema hóspede para o do sistema nativo disputando recursos com outros eventuais sistemas hóspedes. Tecnicamente não há maiores empecilhos em se fazer isso, porém, esse tratamento também representa uma queda de desempenho.

A estratégia de **paravirtualização** (figura 4.8) aparece como uma abordagem alternativa para contornar os problemas de desempenho da virtualização total. Nessa abordagem, o sistema hóspede deve ser modificado para chamar[4] a máquina virtual sempre que for executar uma instrução ou ação considerada sensível. Na prática isso traduz pela necessidade de alterar todas as instruções de sistema (*system* ISA) do hóspede por chamadas a máquina virtual para que ela interprete e emule essas ações de forma adequada. Isso é a principal desvantagem da paravirtualização. A vantagem é que as instruções de usuário (*user* ISA) não precisam ser alteradas e podem ser executadas diretamente sobre o processador nativo. Ao preservar o *user* ISA, todas as aplicações que foram desenvolvidas para o sistema hóspede podem ser executadas sem alterações. Note que, nesse caso, se considera que tanto o sistema hóspede quanto o hospedeiro (sistema nativo) são compilados para a mesma plataforma.

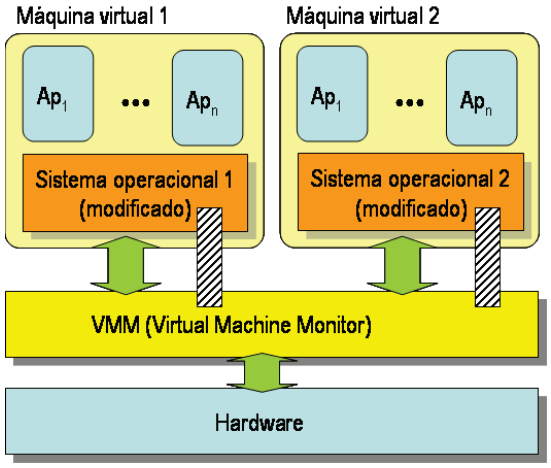

Figura 4.8 – Paravirtualização

Uma outra vantagem da paravirtualização é que ela provê aos sistemas hóspedes acessos aos recursos de hardware a partir dos *drivers* instalados no próprio hipervisor. Dessa forma, os sistemas hóspedes podem usar os recursos reais da máquina e não apenas dispositivos genéricos como era o caso na virtualização total. Ainda, a máquina virtual apenas gerencia o compartilhamento do hardware e monitora as áreas de memória e de disco alocadas para cada um dos sistemas hóspedes. Em decorrência dessa gerência, a máquina virtual pode informar ao sistema operacional hóspede qual a área de memória física que está alocada para ele. Assim, a gerência de memória do hóspede é capaz de traduzir diretamente as páginas virtuais em quadros físicos, sem necessitar de uma etapa adicional de conversão.

[4] O termo normalmente usado é *hypercall*, ou seja, a substituição da chamada de uma instrução sensível pela chamada a um tratador de interrupção de software (*trap*) com os parâmetros adequados.

4.5. Ferramentas de virtualização: exemplos

A virtualização se tornou a grande revolução da área de TI nesses últimos anos, basta ver o crescimento do volume de investimento das empresas nesse sentido e o crescimento das empresas que oferecem soluções de virtualização. Atualmente, existem disponíveis várias soluções de virtualização. Para ter uma idéia dessa quantidade, consulte a *wikipedia,* ou o *google*, fazendo uma busca por "*virtualization*". Basicamente, existem soluções comerciais, gratuitas, em software livre, integradas a sistemas operacionais, etc. Seria inviável, e fora do escopo deste capítulo, tecer comentários sobre todas elas, por isso optou-se por apresentar apenas o Xen, VMware, Virtual PC 2007 e VirtualBox, por serem as mais comuns.

4.5.1 Xen

O Xen (*www.xensource.com*) é um monitor de máquina virtual (hipervisor) em software livre, licenciado nos termos da GNU General Public Licence (GPL), para arquiteturas x86, que permite vários sistemas operacionais hóspedes serem executados em um mesmo sistema hospedeiro. A primeira versão do Xen é de outubro de 2003 e, originalmente empregava a estratégia de paravirtualização, ou seja, era necessário modificar o sistema operacional hóspede para torná-lo consciente da existência do hipervisor. Essa decisão se justificava por questões de desempenho, mas limitou o emprego do Xen aos sistemas Unix, principalmente aqueles com filosofia de código aberto. A partir da versão 3, o Xen passou a oferecer virtualização total, ou seja, permite o uso de sistemas operacionais não modificados, como os da família Microsoft Windows. Entretanto, isso só é possível se o processador oferecer suporte em hardware (Intel VT ou AMD-V) para virtualização.

A arquitetura do Xen é um pouco diferente daquela apresentada na seção 4.4. Os dois principais conceitos do Xen são domínios e hipervisor. Os domínios são as máquinas virtuais do Xen e são de dois tipos: privilegiada (domínio 0) ou não-privilegiada (domínio U). O hipervisor tem por função controlar os recursos de comunicação, de memória e de processamento das máquinas virtuais, e não possui *drivers* de dispositivos. O hipervisor Xen, considerando suas características, não é capaz de suportar nenhum tipo de interação com sistemas hóspedes. Por isso, é necessário que exista um sistema inicial para ser invocado pelo hipervisor. Esse sistema inicial é o domínio 0. As outras máquinas virtuais só podem ser executadas depois que ele for iniciado. As máquinas virtuais de domínio U são criadas, iniciadas e terminadas através do domínio 0. O domínio 0 é uma máquina virtual única que executa um núcleo Linux modificado e que possui privilégios especiais para acessar os recursos físicos de entrada e saída e interagir com as demais máquinas virtuais (domínios U). O domínio 0, por ser um sistema operacional modificado, possui os *drivers* de dispositivos da máquina física e dois *drivers* especiais para tratar as requisições de acesso à rede e ao disco efetuados pelas máquinas virtuais dos domínios U.

Para oferecer suporte tanto para a paravirtualização como para a virtualização total, o Xen distingue os domínios U entre paravirtualizados (domínios U-PV) e virtualizados (domínios U-HVM, de *Hosted Virtual Machines*). Os domínios U-PV têm consciência de que não tem acesso direto ao hardware e reconhecem a existência de outras máquinas virtuais. Os domínios U-HVM não têm essa consciência, nem reconhecem a existência de outras máquinas virtuais. Na prática, isso se traduz no fato de que os domínios U-PV possuem *drivers* específicos para acesso à rede e ao disco para interagirem com as suas contra-partidas no domínio 0. Já as máquinas dos domínios U-HVM não possuem esses *drivers* (não foram modificados) e iniciam como

um sistema convencional procurando executar a BIOS. O *Xen virtual firmware* simula a existência da BIOS fazendo todos os procedimentos esperados durante o *boot* normal de um ambiente PC compatível. O compartilhamento do disco e as requisições de rede de um domínio U-HVM são feitos através de um *daemon* Qemu vinculado a cada instância U-HMV (O QEMU é um emulador em software de código livre). O hardware disponível para as máquinas virtuais do domínio U-HVM são aquelas oferecidas pelo QEMU.

4.5.2 VMware *Player*

A VMware *player* (*www.vmware.com*) é um monitor de máquina virtual que segue a estratégia de virtualização total. Na realidade, por questões de desempenho, o VMware *Player* não executa completamente em espaço de usuário, pois é instalado um *driver* de dispositivo específico, (*VMDriver*) que permite que as máquinas virtuais acessem os *drivers* de dispositivo do sistema hóspede. Por exemplo, o VMDriver põe a placa de rede em modo promíscuo e cria uma *bridge* ethernet virtual que recebe todos os quadros ethernet e os reencaminha para o sistema hóspede ou para a máquina virtual especificada. Essa implementação também oferece NAT (*Network Address Translation*), de tal forma que cada interface virtual tem seu próprio endereço IP. A gerência de memória também é feita de forma híbrida. Cada instância do VMware recebe uma área de memória do sistema operacional nativo e a gerencia diretamente.

A máquina virtual VMware *Player* é disponível gratuitamente para os sistemas operacionais nativos Windows e Linux e existe uma versão comercial (VMware Fusion) para o sistema operacional MacOS X. O objetivo é permitir que usuários tenham contato com a virtualização. A VMware, a partir de seu site, distribui uma série de imagens de sistemas operacionais, denominadas de *appliances*, que contemplam diferentes distribuições linux e Windows Server 2003. A VMware *Player* não permite que um sistema operacional hóspede seja criado na máquina virtual a partir de zero. Para que isso seja possível, dentro das opções oferecidas pela VMware, é necessário uma outra versão, VMware *Workstation*, que é paga.

Na realidade, a VMware oferece uma infra-estrutura de virtualização completa com produtos abrangendo desde *desktops* a *data centers* organizados em três categorias: gestão e automatização, infra-estrutura virtual e virtualização de plataformas. Cada categoria possui um conjunto de produtos específicos. Os produtos de gestão e automatização têm por objetivo principal, como seu próprio nome induz, permitir de uma forma automatizada e centralizada a gerência de todos os recursos da infra-estrutura virtual permitindo a monitoração do sistema, auxiliando na conversão de sistemas físicos em virtuais, na recuperação de desastres, entre outros.

Os produtos de infra-estrutura virtual auxiliam a monitoração e alocação de recursos entre as máquinas virtuais de forma a atender requisitos e regras de negócios. Eles fornecem soluções para alta-disponibilidade, *backup*, migração de máquinas virtuais e atualização de versões de softwares.

Por fim, os produtos de virtualização de plataformas, ou seja, aqueles destinados a criar máquinas virtuais. Essa categoria é composta vários produtos onde se destacam, além dos produtos mencionados anteriormente, o VMware ESX *Server* 3; o VMware ESX *Server* 3i; o VMware Virtual SMP; o VMware VMFS e VMware *Server*, que auxiliam a criação de *data centers* virtuais, exploração de mais de um processador por máquina virtual e a criação de sistemas de arquivos.

4.5.3 Virtual PC 2007

Atenta ao movimento da virtualização, a Microsoft oferece uma gama de produtos para esse tipo de tecnologia. Esses produtos exploram o conceito da virtualização, na sua forma mais ampla, para oferecer soluções que sejam integradas e apropriadas à infra-estrutura de TI que se encontra hoje em dia. Basicamente:

- Virtualização de aplicações: também denominada de *SoftGrid*, cujo objetivo é fornecer aplicações por demanda. Isso implica que se um determinado *desktop* necessita executar uma aplicação e a mesma não está disponível nele, o sistema executará automaticamente a busca, instalação e configuração da aplicação.
- Virtualização de apresentação: essa ferramenta separa e isola as tarefas de tratamento gráfico (visualização) e de E/S, permitindo que uma determinada aplicação seja executada em uma máquina, mas utilize recursos gráficos e de E/S de outra.
- Gerenciamento da virtualização: *System Center Virtual Machine Manager* é um ambiente de gerenciamento que facilita as tarefas de configuração e de monitoração de um ambiente virtual. É através dele que se realiza a administração das contas de usuários e seus privilégios.
- Virtualização de *desktops* (Virtual PC): permite a criação de máquinas virtuais em um sistema hospedeiro Microsoft Windows, cada uma com seu próprio sistema operacional. Basicamente destina-se àquelas aplicações onde é necessário executar software legado, criar ambientes de testes, realizar treinamentos, etc.
- Virtualização de servidores: é a solução que permite criar máquinas virtuais em servidores. Nessas máquinas, questões ligadas à segurança, tolerância a falhas, confiabilidade, disponibilidade se tornam importantes. Portanto, a solução de virtualização de servidores, denominada de Hyper-V (ou *viridian*) foi projetada para endereçar esses requisitos.

Dentre esses produtos, o que se enquadra na discussão desta seção é o virtual PC 2007 para virtualização de *desktops* (*www.microsoft.com/Windows/virtualpc*). O Virtual PC 2007 é uma máquina virtual para família Windows que pode ser configurada para executar qualquer outro sistema operacional. A estratégia adotada é a da virtualização total.

Segundo a Microsoft, o principal objetivo do Virtual PC é o desenvolvimento e teste de software para múltiplas plataformas. Dentro desse princípio, o Virtual PC oferece mecanismos para interconectar logicamente as diferentes máquinas virtuais. Cada máquina virtual tem seu próprio endereço MAC e endereço IP. Além disso, o Virtual PC oferece um servidor de DHCP, um servidor NAT e *switches* virtuais. Dessa forma, é possível construir cenários de rede usando máquinas virtuais. O virtual PC 2007 é disponível para *download*, assim como um *white paper* que ensina a configurar as máquinas virtuais e um ambiente de rede. Um ponto interessante a comentar em relação à gratuidade do Virtual PC é que, na FAQ do Virtual PC, a Microsoft alega que o que tem valor agregado não é a máquina virtual em si, mas sim os ambientes de gerenciamento. Uma diferença que o Virtual PC 2007 tem em relação ao VMware é a possibilidade de se definir máquinas virtuais e instalar o seu próprio sistema operacional. É importante salientar que mesmo em uma máquina virtual as restrições de licenças são aplicadas.

4.5.4 VirtualBox

O VirtualBox (*www.virtualbox.org*) é um monitor de máquina virtual que adota a abordagem de virtualização total. O VirtualBox é parte integrante de sua solução de virtualização da Sun Microsystems, o Sun xVM.

Existem duas versões do VirtualBox: uma versão em software livre, sob os termos da GPL (GNU *General Public Licence*) denominada de VirtualBox OSE (*Open Source Edition*) e outra comercial (versão *full*). A diferença básica entre elas é o suporte que a versão comercial oferece para RDP (*Remot Desktop Protocol*), para USB e para i-SCSI (mecanismo de acesso a discos SCSI remotos). O VirtualBox executa sobre sistemas operacionais Microsoft Windows, Linux, Macintosh e OpenSolaris permitindo vários sistemas hóspedes, entre eles, os mais comuns, Microsoft Windows (NT, 2000, XP, Vista e Server 2003), GNU/Linux e OpenBSD, entre outros.

A virtualização feita pelo VirtualBox executa todas as instruções de usuário (*user* ISA) nativamente no processador. Isso, é claro, se houver compatibilidade binária entre o sistema hóspede e o hospedeiro. As instruções de sistema (*system* ISA) são interceptadas e emuladas no hospedeiro. Essa emulação é feita com o auxílio da técnica de tradução binária dinâmica.

Ainda, o VirtualBox emula um disco virtual com um arquivo em um formato específico, o *Virtual Disk Image*, mas possui a capacidade de ler e escrever imagens de discos virtuais VMware (*Virtual Machine Disk Format* – VMDK). Isso possibilita que o VirtualBox possa ser inicializado a partir de uma imagem definida e criada por ferramentas da VMware. Outra característica interessante no VirtualBox é o fato da permitir a montagem de imagens ISO. Assim, é possível, por exemplo, usar uma imagem de uma distribuição linux sem ter que "queimar" um CD/DVD

4.6. Exemplos de Emprego da Virtualização

Como já comentado anteriormente, as máquinas virtuais oferecem um ambiente completo, cada uma contendo seu próprio sistema operacional, bibliotecas e aplicativos de uma forma totalmente auto-contida e independente. Essa característica pode ser explorada de forma a trazer uma série de benefícios para diferentes infra-estruturas de computação. As aplicações típicas de máquinas virtuais são a consolidação ou virtualização de servidores, virtualização de *desktops* e segurança (*honeypots*). O emprego da virtualização em ambientes de processamento paralelo e distribuído é um exemplo de uso dessas aplicações típicas, mas em um outro contexto. Nesta seção iremos elaborar um pouco mais cada uma dessas aplicações.

4.6.1 Virtualização de servidores

Hoje em dia é muito difícil imaginar um sistema computacional que não seja conectado em rede. Na prática, essa conectividade faz com que os administradores de rede sejam responsáveis por manter um conjunto grande e heterogêneo (hardware e/ou softwares distintos) de servidores, cada um executando uma aplicação diferente que pode ser acessada por clientes também heterogêneos.

É comum encontrarmos em infra-estruturas de rede uma filosofia "um servidor por serviço" por motivos que variam desde a heterogeneidade de clientes à segurança. Normalmente, a carga de processamento de um servidor não explora todo o poder computacional disponibilizado pelo processador. Há um desperdício de ciclos de processamento e, por conseqüência, de investimento. Nessa situação, pode-se imaginar cada serviço executando em uma máquina virtual individual, com seu próprio sistema

operacional, mas sobre um mesmo hardware físico, proporcionando um uso eficiente do poder de processamento disponível. Isso é o que se denomina de consolidação de servidores. O princípio básico é o melhor aproveitamento de recursos: ao invés de haver *n* servidores com percentual de utilização de *x* é possível ter um único servidor com um percentual de uso de aproximadamente *n.x* (*n.x* < 100). A consolidação de servidores contribui para a diminuição de máquinas físicas, o que auxilia na redução de custos de infra-estrutura como espaço, energia elétrica, cabeamento, refrigeração, suporte e manutenção de vários sistemas.

Inicialmente, vamos discutir o caso de pequenas e médias organizações e seus servidores. Nesse contexto, é comum que os servidores sejam máquinas ultrapassadas e "intocáveis". Afinal, se os serviços executados não excedem a capacidade dessas máquinas, por que investir em equipamentos novos? Além disso, atualizar máquinas implica em mexer em sistemas que, bem ou mal, estão funcionando. Entretanto, essa situação traz um risco inerente: um hardware está sujeito a falhas, principalmente os discos, e, muitas vezes, para equipamentos ultrapassados, os contratos de manutenção não são mais válidos e as peças de reposição simplesmente inexistem. É uma questão de tempo ter um problema (grave) de indisponibilidade de serviços.

Para esses casos, uma solução possível é adquirir apenas um único equipamento novo e instalar nele tantas máquinas virtuais quanto serviços houver. É claro que se deve dimensionar a capacidade total de processamento desse novo servidor em relação à carga de serviços que ele receberá. A vantagem é trocar um parque de máquinas ultrapassadas e, eventualmente, subutilizadas, por uma máquina mais nova e bem utilizada. Os serviços que executam nas máquinas mais antigas podem ser migrados, um a um, para uma máquina virtual na nova máquina, facilitando o processo de transição. Dependendo da solução de virtualização escolhida, existem ferramentas que auxiliam nessa migração. A notar, ainda, que cada máquina virtual tem seu próprio sistema operacional, portanto, é possível pôr em um único equipamento todos os serviços, ou aplicações, que executam em diferentes sistemas operacionais. Futuramente, se for necessário uma nova atualização de hardware, basta parar as máquinas virtuais, fazer uma imagem delas e as inicializar novamente, sem necessitar reinstalar e configurar softwares específicos.

Um segundo caso a analisar são os *data centers*, onde as vantagens discutidas anteriormente tomam uma envergadura maior. Por sua própria natureza, em um *data center* é comum haver diferentes sistemas operacionais e, eventualmente, mais de uma versão de um mesmo sistema operacional devido a necessidades específicas de produtos que seus clientes executam. Sem virtualização, cada sistema operacional (e versão) precisaria de um hardware dedicado. Isso representa um custo de instalação, manutenção, refrigeração, energia e suporte técnico para administrá-los. Já com a virtualização, um único hardware, dimensionado adequadamente, pode manter vários sistemas operacionais de forma mais econômica.

Os procedimentos de migração e de tolerância a falhas são importantes em *data center*. A migração pode ser usada no momento de atualizar parques de máquinas movendo um serviço de uma máquina para outra enquanto a troca é feita. A migração ainda é útil como mecanismo de balanceamento de carga, pois permite mover serviços de uma máquina a outra até que a carga do sistema seja homogênea. Isso evita que certos sistemas fiquem sobrecarregados enquanto outros estão ociosos.

4.6.2 Virtualização de *desktops*

A virtualização também traz uma série de benefícios quando empregada em *desktops*. Inicialmente, a virtualização oferece uma forma simples para testar novas configurações ou executar programas que foram feitos para sistemas operacionais diferentes do nativo. Dessa forma, um usuário que deseja testar um software, ou abrir um programa, que não foi desenvolvido para seu sistema operacional pode lançar mão desse recurso. Há uma série de ferramentas para o uso de virtualização em *desktops*, onde entre elas, se destacam os produtos da VMmare e o Virtual PC por serem de fácil instalação e uso.

Os projetistas de software são um tipo de usuário que podem se beneficiar ao executar programas de diferentes sistemas operacionais em uma única máquina física. As máquinas virtuais permitem isso de maneira bastante simples, o que é interessante em fases de desenvolvimento e depuração de software. Um outro ponto, vinculado à depuração de sistemas, é a facilidade que as máquinas virtuais têm de registrar uma espécie de instantâneo (*snapshot*) de seu estado quando finalizadas. Isso permite que uma análise seja parada em qualquer instante e retomada mais tarde a partir do mesmo ponto.

Outra utilização interessante da virtualização de *desktops* é em laboratórios de treinamento e de ensino, como aqueles encontrados em universidades. Para algumas disciplinas é necessário ter ambientes com diferentes sistemas operacionais. Apesar de ser possível, e comum, se configurar um PC compatível com múltiplos *boots*, essa solução nem sempre é a mais adequada. Primeiro, dependendo dos sistemas operacionais, questões práticas afetam a ordem de instalação e de configuração dos diversos sistemas. Segundo, e talvez mais importante, é comum haver cursos e treinamentos onde o participante necessita ter acessos administrativos. Nesses casos, mesmo mudando as senhas de administrador (*root*) a cada sessão, há a possibilidade de se corromper o sistema operacional pela instalação de *rootkits, backdoors, sniffers* de teclado, e todo um conjunto de *malwares,* além de ser possível, por exemplo, montar a partição dedicada a um outro sistema operacional e provocar danos. Nesse caso, as máquinas virtuais são bem apropriadas: o participante de uma disciplina pode usar um ambiente virtual para fazer todos os experimentos e, por rodar de forma confinada em um processo, suas ações como administrador não afetam o sistema nativo. Ao finalizar o processo (máquina virtual) não há nenhum vestígio permanente do que foi feito. Um detalhe, havendo necessidade de preservar arquivos de uma aula a outra, as máquinas virtuais oferecem formas de comunicação (*ftp, telnet, scp,* etc) e armazenamento (*usb*) que possibilitam a cópia e o salvamento de arquivos.

4.6.3 Honeypots

De uma forma simplista, ***honeypot*** consiste em se colocar intencionalmente máquinas na Internet de forma que elas sejam atacadas por *crackers*. O intuito é monitorar as atividades desses, se precaver de ataques e tornar mais fácil a investigação de incidências de ataques e de sua recuperação. O problema com *honeypots* é que, dependendo do tipo de ação sofrida, pode haver o comprometimento da máquina (sistema operacional). Nesses casos, a solução passa por reinstalar o sistema. Outra característica do uso de *honeypots* é que eles normalmente são compostos por máquinas destinadas a essa finalidade e que são postas em segmentos de redes específicos, o que aumenta os investimentos tanto em nível de hardware como de suporte de administração de rede.

A virtualização surge como uma opção interessante para se implementar *honeypots* por várias razões. Primeiro, o comprometimento de um sistema operacional é

resolvido apenas com a remoção da máquina virtual e a instanciação de uma nova máquina virtual. Uma segunda razão é que, em um mesmo hardware físico, é possível instalar várias máquinas virtuais, cada uma com um sistema operacional diferente, o que permite aumentar o número de "máquinas iscas". Por fim, com o uso de softwares de emulação de equipamentos de rede, pode-se definir uma infra-estrutura de rede virtual, com *firewalls*, *proxies*, roteadores etc, em uma única máquina. Note que isso não é imprescindível, pois as ferramentas de virtualização oferecem normalmente suporte a interfaces de redes virtuais, com endereços MAC e IP distintos, e a VLANs, o que permite interconectar as máquinas virtuais em equipamentos de interconexão físicos sem maiores problemas.

Uma outra situação, associada com a idéia de *honeypot*, é a possibilidade de se executar softwares de origens "não confiáveis" em um ambiente onde os prejuízos são minimizados. Isso é interessante para abrir arquivos *attachment* suspeitos ou verificar *malwares*. Vale a pena salientar que as ferramentas tradicionais existentes para *honeypots* podem ser utilizadas sem restrições alguma em ambientes virtualizados, pois elas são aplicações para um sistema operacional. Uma boa referência para ferramentas de *honeypots* é *www.honeyclient.org*.

4.6.4 Processamento Paralelo e Distribuído

Uma característica importante das máquinas virtuais é a sua capacidade de oferecer um ambiente computacional completo, isso é, sistema operacional, bibliotecas, arquivos de configurações e aplicações em uma imagem (tipo especial de arquivo). Uma máquina virtual pode então ser vista como um aplicativo que recebe como parâmetro um arquivo de imagem. Isso se traduz na possibilidade de se definir um ambiente de trabalho e criar uma espécie de "pacote de distribuição" a ser instalado e usado em qualquer máquina. Outro aspecto muito interessante das máquinas virtuais é a sua capacidade de isolamento, ou seja, o que acontece no interior de uma máquina virtual não afeta o sistema nativo, nem outras máquinas virtuais. Além disso, uma máquina virtual pode ser parada em qualquer momento preservando seu estado atual o qual pode ser recuperado posteriormente. Esses aspectos fornecem perspectivas interessantes para o uso da virtualização em ambientes de processamento paralelo e distribuído que serão vistos a seguir.

Infra-estrutura de processamento paralelo: Já há muito tempo *clusters* e *grids* se tornaram uma infra-estrutura para processamento paralelo e distribuído por fornecerem um poder computacional semelhante ou superior ao de supercomputadores com um custo significativamente mais baixo. Grosso modo, ambos podem ser vistos como um conjunto de máquinas independentes interconectadas em rede e destinadas a execução de uma aplicação paralela. Uma diferença fundamental entre esses dois sistemas é o fato de que os *clusters* pertencem a um mesmo domínio administrativo ao passo que um *grid* é formado por vários deles. Por domínio administrativo entende-se uma organização que é responsável por gerenciar e manter os recursos de um ambiente de rede. Isso implica em instalação de sistemas operacionais e softwares (bibliotecas, aplicativos etc), configuração, manutenção das máquinas e cadastro de usuários. Existem dois problemas essenciais com *clusters* e *grids*: a grande quantidade de máquinas que podem constituir um ou outro e que devem ser corretamente instaladas, configuradas e mantidas; e a segurança que atinge principalmente os *grids* já que envolve vários domínios administrativos. Inicialmente, vamos considerar o caso de *clusters*.

Um *cluster* pode ser formado por centenas de máquinas. É impraticável executar tarefas administrativas nessa quantidade de máquinas de forma manual, portanto muitos

esforços foram feitos no sentido de automatizar as tarefas de instalação, configuração e manutenção de sistemas baseados em *cluster*. Talvez o mais significativo deles tenha sido o Oscar [OSCAR 2008]. A questão de segurança é de certa forma minimizada já que um *cluster* é tipicamente formado por uma máquina frontal (*front-end*) conectada a duas redes: uma rede acessível a partir do exterior e uma rede interna formada pelos nós de processamento. Para lançar uma aplicação, um usuário se autentica na máquina frontal e a partir dela dispara a execução de sua aplicação. Assim é relativamente fácil gerenciar a segurança já que basicamente garantir acesso seguro ao *front-end* aos usuários com conta nesse ambiente.

No entanto, os *clusters* têm duas limitações inerentes a sua construção. Primeira, em um ambiente de *cluster* todas as máquinas possuem um mesmo sistema operacional. Segunda, eventualmente, devido à velocidade do mercado de computadores, não se consegue aumentar, de forma homogênea, o número de máquinas de um *cluster* construído no passado por não se encontrar mais exatamente o mesmo tipo de hardware. Tem-se então um problema de heterogeneidade que pode ter impactos em relação ao conjunto de software instalado. Um *cluster*, quanto mais homogêneo for, melhor é sua manutenção.

Nesse ponto, a virtualização surge como uma solução com uma série de vantagens. A primeira delas é a capacidade de fazer com que o *cluster* seja visto como um conjunto de máquinas virtuais. Sendo assim, a homogeneidade do sistema é dada pelo fato de que as máquinas virtuais são instanciadas a partir de uma mesma imagem independente do hardware e do sistema operacional sobre as quais executam. Segundo, um *cluster*, como visto anteriormente, é fisicamente formado por um *front-end* e por um conjunto de nós de cálculo. Isso pode representar um desperdício de investimento dependendo da taxa de utilização do *cluster* e de seus nós. Porque então não criá-los e dimensioná-los a medida do necessário, por demanda? Por fim, a instalação, configuração e manutenção de um *cluster* são feitas a partir de uma única imagem, a qual posteriormente é usada em todos os nós. Considerando essas facilidades foram desenvolvidos alguns projetos voltados a *clusters* usando a virtualização como, por exemplo, o OSCAR-V [Vallée et al, 2007] e *Cluster on Demand* (COD) [Emeneker et al. 2006]. O primeiro é uma versão estendida de seu predecessor OSCAR que adiciona a capacidade de criar e gerenciar máquinas virtuais sobre um *cluster* físico. No segundo, a criação de um *cluster* é feita por demanda, inicialmente localizando quais máquinas físicas (recursos) estão disponíveis. Na seqüência, é feita a instanciação de uma máquina virtual sobre cada um deles e a criação de um contexto. O contexto pode ser visto, por exemplo, como a configuração da quantidade de nós disponíveis para se lançar uma aplicação MPI.

Um *grid* computacional pode ser visto, de certa forma, como a extensão natural de um *cluster*. Os *grids* estendem a noção de máquina virtual para aquela de uma organização virtual que permite o compartilhamento de recursos (processamento, memória, discos, dados etc) entre sistemas que estão geograficamente dispersos. Uma organização virtual pode ser vista como uma coleção de usuários e recursos, construída sobre um conjunto de integrantes de diferentes administrativos e controlada por regras de compartilhamento. Em um *grid* os problemas de instalação, configuração e manutenção de um grande parque de máquinas apresentam um fator de complexidade adicional por atravessar domínios administrativos e implicar em regras e restrições de segurança diferentes. Para sintetizar essa questão, imagine a reação de um administrador de redes de uma organização ao receber o pedido, de um usuário que ele nunca ouviu falar, de que é preciso instalar uma biblioteca no sistema para seu software executar?

A idéia fundamental da computação em *grid* é que para lançar uma aplicação um usuário descreve os recursos necessários. A partir disso, os recursos são localizados e alocados e, finalmente, a aplicação é lançada. Entretanto, isso não é simples assim. Primeiro, nem sempre é fácil se obter um ambiente adequado para se executar uma dada aplicação. Além de encontrar os recursos mais apropriados é necessário, muitas vezes, negociar o seu uso. Existem vários fatores que afetam os critérios de escolha de recursos, por exemplo, sites diferentes oferecem diferentes requisitos de qualidade de serviço ou, ainda, necessidade de realizar experimentos sazonais exigindo "picos" de processamento. Na prática, o problema maior não são os recursos, pois é relativamente fácil uma organização possuir um determinado número de máquinas ou de *cores*. As questões principais são: como instalar e manter os pacotes de softwares necessários? Como conciliar os conflitos entre usuários e as restrições de segurança do site? Como coordenar o escalonamento de aplicações?

Uma possibilidade de resposta a essas questões é a criação de ambientes virtuais de trabalho, ou v*irtual workspaces* [Keahey et alli, 2005]. Essencialmente, a idéia consiste em alocar dinamicamente recursos e configurá-los de forma a oferecer para o usuário um ambiente computacional completo por demanda. Por recursos, se entende a necessidade de processamento (cpu), de memória, de disco, de banda passante e que, esses recursos possam ser renegociados mesmo durante a execução da aplicação de forma a refletir mudanças de requisitos e restrições. Nesse contexto, denomina-se de *appliance*, o ambiente de software necessário para uma aplicação.

O projeto *Virtual Workspace* evoluiu de tal forma que hoje ele é apenas um módulo de uma infra-estrutura mais completa, o Nimbus. O projeto Nimbus [Virtual, 2008] é uma iniciativa de empregar a tecnologia de virtualização para computação em *Grid*. Basicamente, o Nimbus é um conjunto de ferramentas que permite que se descreva um conjunto de requisitos necessários a uma infra-estrutura e, de forma automatiza, é feita a procura e a configuração de recursos (máquinas) que possam atender esses requisitos. Os conceitos originais do Nimbus são do projeto *Virtual Workspaces*. As principais tecnologias empregadas no Nimbus são o Xen para máquinas virtuais e o uso de *web services*, através do WSRF (*Web Services Resource Framework*) para o suporte a descrição e localização de recursos. Além do Nimbus, existem outros projetos que buscam a definição de uma infra-estrutura baseada em máquinas virtuais para implementar um grid computacional. Entre outros, é possível destacar o Virtuoso [Virtuoso, 2008] e o VSE [Engelmann, 2007].

Balanceamento de carga e tolerância a falhas: Como mencionado anteriormente um dos aspectos interessantes das máquinas virtuais é que elas encapsulam um ambiente computacional completo em qualquer instante de tempo. Essa característica permite que se suspenda a execução de uma máquina virtual e que se preserve o seu estado atual de execução criando uma espécie de "fotografia instantânea" do ambiente. Esse estado pode ser posteriormente restaurado permitindo que a execução da máquina virtual siga a partir do ponto em que ela havia sido suspensa. Em uma infra-estrutura de rede, essa facilidade permite que uma máquina virtual seja parada, transferida para outra máquina física, e reiniciada, o que é particularmente útil para se fazer a manutenção ou *upgrade* em uma máquina física sem causar a indisponibilidade de serviços. Isso é o que se denomina de migração. A migração pode ser empregada como um mecanismo de base para balanceamento de carga e de tolerância a falhas, que são extremamente úteis em processamento paralelo.

Na prática, o encapsulamento feito por uma máquina virtual guarda muitas similaridades com um mecanismo de *checkpointing*/*restart*. Ele permite que uma máquina virtual seja suspensa por um tempo indeterminado e que se retome a execução exatamente no ponto em que a suspensão foi feita. Se a suspensão e a retomada forem feitas em máquinas diferentes, tem-se a migração. Se o *checkpoint* de uma aplicação executando sobre uma ou mais máquinas virtuais é feito regularmente é possível preservar estados conhecidos e coerentes dessa aplicação. Isso é particularmente interessante para aplicações que executam durante muito tempo ou em ambientes em que os recursos podem ser tornar indisponíveis, como o caso de *grids*. Nessa situação, em caso de pane ou indisponibilidade de recursos, é possível retomar a execução a partir do último *checkpoint* realizado com sucesso, evitando a perda do processamento executado.

No entanto, apesar de conceitualmente simples, essa abordagem traz consigo alguns desafios. Primeiro, em *cluster* e *grids*, há o problema de que uma aplicação executa em nós distintos que se comunicam, na grande maioria das vezes, através de conexões TCP. A mudança de um endereço IP e das portas usadas na conexão fazem com que a mesma seja "quebrada". O segundo desafio é relacionado com o tamanho da imagem. Dependendo da aplicação que está sendo executada, e do ambiente necessário para tal, é comum uma imagem ocupar centenas de megabytes a alguns gigabytes. Nesse caso, para migrar uma aplicação de um nó a outro, é necessário realizar uma transferência de imagem via rede onde a latência e a banda passante se tornam fatores importantes.

O problema da troca do endereço IP é relativamente simples de ser contornado através do uso de técnicas como IPMobile, VPN e NAT para definir uma rede virtual com endereços IP independentes do endereço IP da máquina hospedeira. Portanto, a aplicação executa sempre usando o mesmo IP (virtual). Já o tamanho da imagem é um pouco mais delicado de ser tratado e duas abordagens são utilizadas: reduzir o tempo de transferência da imagem pela rede ou decidir se a vale a pena ou não realizar uma migração.

Uma primeira abordagem para reduzir o tempo de transferência é a utilização de um sistema de arquivos distribuídos [Kozuch, Satyanarayanan 2002]. Nesse esquema, todas as máquinas compartilham um mesmo sistema de arquivos via rede, mas a máquina alvo faz a leitura do arquivo imagem de forma incremental, por demanda, e em *background*. Assim, não é necessário esperar a transferência ser concluída para iniciar a execução da máquina virtual. Nesse trabalho, há ainda a sugestão de eventualmente reaproveitar porções da imagem que já estejam na máquina alvo. Isso é válido para qualquer tipo de arquivo no interior da imagem que seja *read-only*. Uma variação da abordagem anterior é proposta no projeto *Stanford Collective* [Sapuntzakis et al, 2002]. Nesse trabalho, foi feita uma série de otimizações para reduzir a quantidade de dados a serem transferidos de uma máquina a outra através de técnicas como compactação, reaproveitamento de blocos de disco etc.

Uma vez que há condições para migrar máquinas virtuais de forma eficaz é possível usar a migração como um mecanismo de base para executar balanceamento de carga. Assim, um sistema pode ficar monitorando a carga das diversas máquinas e ao detectar a ociosidade de uma em relação à outra é possível deslocar carga de processamento de uma máquina a outra. Entretanto, é necessário que o custo de migração (*checkpointing*, transferência e *restart*) sejam menos onerosos que o que se espera ganhar com a migração. Também vale a pena ressaltar que a granularidade da migração é de uma máquina virtual completa, portanto, migrar uma máquina virtual de uma máquina física para outra só se justifica se a máquina sobrecarregada estiver executando mais de uma máquina virtual ou se a máquina ociosa tiver uma capacidade de processamento muito superior. Em [Righi et al., 2008] é proposto um modelo de custo para dimensionar a viabilidade de uma migração. Ainda, em [Veiga et al, 2007] é apresentado o custo de migração de máquinas virtuais em um trabalho baseado no Xen.

4.7. Considerações finais

Até o presente momento a virtualização foi colocada como uma solução para vários problemas, no entanto, ela também apresenta alguns inconvenientes quando a aspectos de segurança, gerenciamento e desempenho.

Inicialmente, o hipervisor é uma camada de software e, como tal, está sujeito a vulnerabilidades. Segundo Neil MacDonald, especialista de segurança da Gartner, hoje em dia, as máquinas virtuais são menos seguras que as máquinas físicas justamente por causa do hipervisor. Mas há muita controvérsia nesse campo. Além disso, há a questão da disponibilidade de serviços que ocorre com o comprometimento, lógico ou fisico, da máquina física que hospeda vários servidores virtuais, já que todos seriam afetados simultaneamente. É verdade que as soluções corporativas, como o ESX *Server* e o Citrix Enterprise, permitem toda uma monitoração dos sistemas, replicação, migração e *backup* dos sistemas virtualizados, que diminuem o tempo de recuperação em caso de problemas, mas é algo a ser avaliado. Um outro argumento freqüentemente empregado em favor da disponibilidade é que um *pool* de máquinas virtuais é similar a um *rack* com vários servidores físicos, com a vantagem de ter uma flexibilidade e uma portabilidade maior que estes últimos.

O segundo aspecto diz respeito ao gerenciamento das máquinas virtuais. Os ambientes virtuais necessitam serem criados, monitorados, configurados, mantidos e salvos. Existem produtos, como os brevemente citados na seção 4.5, que integram essas soluções, porém ainda há espaço para melhorias. Em especial, há o aspecto de gerenciamento relacionado com a segurança que ainda é deficiente nas atuais soluções: a correlação de eventos e ações feitas como *root* em ambientes virtuais.

Por fim, mas não menos importante, a questão desempenho que pode ser formulada de duas formas. Primeira, qual o custo de processamento introduzido pela camada de virtualização? Segunda, quantas máquinas virtuais são possíveis sem comprometer uma qualidade de serviço (escalabilidade)? Para responder esses questionamentos, vários estudos foram feitos e se encontram disponíveis na Internet, porém, muitos deles são específicos a uma determinada situação ou solução.

Atento a essa necessidade, e como forma de sistematizar e fornecer uma metodologia padrão para avaliar o desempenho de máquinas virtuais, foi criado um comitê especial: o SPEC *Virtualization Commitee* (*www.spec.org/specVirtualization*). O objetivo desse comitê é desenvolver *benchmarks* para avaliar o desempenho da virtualização em servidores e prover mecanismos para comparar o comportamento desses quando executando muitos hipervisores. No momento da redação deste trabalho[5],

[5] Fevereiro de 2009.

a liberação para primeira versão de *benchmark* estava prevista para o primeiro semestre de 2009.

Enquanto não há um *benchmark* específico para avaliar a virtualização, os estudos feitos até agora costumam empregar *benchmarks* existentes que simulam cargas de processamento, de entrada e saída, e de tráfego na rede. A partir de uma avaliação feita pela VMware [VMware 2008a] e posteriormente questionado e reconduzido pela Xen Source [XenSource2008a], se popularizaram os *benchmarks* SPECcpu2000, focado em aplicações computações intensivas; o Passmark, que gera uma carga de trabalho para testar os principais subsistemas que compõem um sistema operacional; o NetPerf, para avaliar o desempenho no envio e recepção de dados via rede; o SPECjbb2005 que representa um servidor e sua carga; e a compilação do pacote SPECcpu2000 INT. Os resultados obtidos pela VMware e XenSource apontaram para uma queda de desempenho, em geral, entre 2% e 10%, com algumas situações impondo perdas maiores. Cabe ressaltar que esses resultados foram obtidos usando *benchmarks* genéricos destinados a outras finalidades. Em [Charão, 2008] é apresentado um outro estudo sobre o desempenho do Xen.

8 Conclusões

Virtualização é o conceito que fornece a abstração de um recurso computacional qualquer mascarando suas características físicas dos usuários e aplicações que os utilizam. No entanto, a forma mais conhecida de virtualização são as máquinas virtuais. Cada máquina virtual, por si só, é um ambiente de execução isolado e independente das demais. Com isso, cada máquina virtual pode ter seu próprio sistema operacional, aplicativos e serviços de rede (Internet). O sistema operacional do hóspede pode ser diferente daquele utilizado pelo hospedeiro. A virtualização não é um conceito recente, remonta a década de 60, e existem várias técnicas para implementá-la, onde se destacam as máquinas virtuais de processo e o monitor de máquina virtual.

Uma máquina virtual é implementada considerando dois pontos de vistas diferentes: a de um processo e a do núcleo do sistema operacional. Uma máquina virtual de processo fornece um ambiente de execução para uma única aplicação através de uma interface virtual (ABI) a qual é mapeada para uma interface real composta pelas chamadas de sistema e pelo conjunto de instruções não-privilegiadas (*user* ISA). Na prática, isso pode ser visto como um processo (a aplicação) executando dentro de outro processo (a máquina virtual). Uma máquina virtual de processo só existe enquanto a aplicação estiver em execução. Uma máquina virtual de sistema, também denominada de VMM (*Virtual Monitor Machine*) ou hipervisor, é aquela que oferece um ambiente de execução completo onde coexiste um sistema operacional e várias aplicações. Ao contrário das máquinas virtuais de processo, as máquinas virtuais de sistema estão ativas até serem explicitamente paradas. Há dois tipos de hipervisores: nativos (tipo I) e hóspedes (tipo II). Os hipervisores nativos são uma camada de software posta entre o hardware físico da máquina e as máquinas virtuais executam sobre eles. Já os hipervisores hóspedes (tipo II) executam sobre o sistema operacional como se fossem um processo deste. Há ainda, duas técnicas usadas nos hipervisores: virtualização total e paravirtualização. A diferença essencial é se o sistema operacional hóspede precisa ser modificado (paravirtualização) ou não (virtualização total) para executar sobre o hipervisor.

Assim como já aconteceu no passado com a multiprogramação, o projeto dos processadores mais recentes tem considerado mecanismos em hardware para dar suporte a virtualização. É o caso dos fabricantes AMD e Intel que desenvolveram extensões para a arquitetura x86 suportar virtualização, respectivamente, AMD-Virtualization (AMD-V, codinome *Pacífica*) e Intel *Virtualization Technology* (IVT, condinome *Vanderpool*).

As principais vantagens da virtualização são:

- Isolamento de dados e de processamento permitindo que uma mesma máquina física execute várias máquinas virtuais sem que elas interfiram umas nas outras e no próprio sistema nativo.
- Capacidade de instalar de maneira rápida e uniforme, independente de plataforma, um mesmo ambiente computacional (sistema operacional, biblioteca e aplicações). Isso permite se ter um ambiente capaz de ser executado em qualquer lugar.
- As máquinas físicas não precisam ter um mesmo hardware para executar uma mesma máquina virtual. Essa característica é uma forma de contornar questões de heterogeneidade. Portanto, as máquinas virtuais são uma forma de se manter ambientes homogêneos sobre hardwares heterogêneos.
- Possibilidade de integrar facilmente mecanismos de migração e de balanceamento de carga através da facilidade de se executar a suspensão e a sua retomada (mecanismo *suspend*/*resume*). Essa mesma facilidade pode ser usada como base para tolerância a falhas através de mecanismos de *checkpoint*/*restart*.

Essas vantagens fazem da virtualização uma ferramenta interessante para ser usada em infra-estruturas de TI e, em ambientes de processamento paralelo, na implementação de *clusters* e *grids*. O uso típico da virtualização em infra-estruturas de TI é na consolidação de servidores, isso é, permitir que vários servidores executem simultaneamente em um único hardware físico, mas cada um em sua própria máquina virtual. Entretanto a virtualização é empregada com sucesso em várias outras situações como ambientes de desenvolvimento e teste de produtos, laboratórios de treinamento de cursos de redes e de sistemas operacionais, e servir de base para implantação de mecanismos de segurança (*honeypots*).

No caso de processamento paralelo e de alto desempenho (PAD) a virtualização surge como uma possibilidade promissora para resolver problemas relacionados com gerenciamento de recursos e migração. Um exemplo são os *clusters* que podem ser dimensionados por demanda e não mais vinculados à existência de hardware físico. Nesse caso, os nós do cluster são máquinas virtuais que podem ser instanciadas em uma ou várias máquinas físicas.

Em ambientes de *grids*, a virtualização encontra um nicho natural, pois um *grid* nada mais é que a definição de um grande sistema virtual. Nesse contexto a virtualização auxilia na resolução de problemas relacionados com a segurança e tolerância a falhas. Primeiro, a partir do instante que uma máquina virtual é uma imagem que pode ser instanciada e executada sobre um sistema operacional nativo sem interferir em seu funcionamento, muito das restrições e cuidados atuais de compartilhar recursos são minimizados. A única exigência que se faz é que a máquina alvo tenha instalado a máquina virtual. Em *grid* a volatilidade dos recursos é algo problemático por representar a perda de computação feita até um momento e para contornar esse problema normalmente são empregados mecanismos de tolerância a falhas. As máquinas virtuais, com sua facilidade de *suspend*/*resume* auxiliam na implementação de tais mecanismos.

Por fim, a virtualização é uma ferramenta muito poderosa e que oferece uma gama de oportunidades, mas ainda existem vários pontos a serem melhores tratados, como as questões de gerenciamento, segurança e de desempenho. Para reforçar essa afirmação uma citação interessante da Microsoft (em seu site *technet*): “o valor de mercado da virtualização não está na pilha necessária para implementá-la, isso é, no

conjunto sistema operacional mais o hipervisor, mas sim no desenvolvimento de sistemas de gerenciamento para ambientes virtuais".

4.9. Referências

Adamic, A. L. "Zipf, Power-laws and Pareto – a ranking tutorial". Information Dynamics Lab, HP Labs. (http://www.hpl.hp.com/research/idl/papers/ranking), acesso fevereiro 2008.

Adams, K.; Agesen, O. "A comparaison of software and hardware techniques for x86 virtualization", ASPLOS'06, San Jose, California, USA. 2006.

Baratz, A.; "Virtual Machines shootout: VMWare vs. Virtual PC", Ars Technica. August, 2004 (http://arstechnica.com/reviews/apps/vm.ars) Acesso novembro 2007.

Barham, P.; Dragovic, B.; Fraser, K.; Hand, S.; Harris, T.; Ho, A.; Neugebauer, R.; Pratt, I.; Warfield, A. "Xen and the Art of Visualization". In. Proc. 19th ACM Symp. On Operating System Principles (SOSP'03), 2003.

Breslau, L.; Cao, P.; Fan, L. ; Philips, G. And Shenker, S. "Web Caching and Zipf-like Distributions: Evidence and Implications". INFOCOM, 1999, pp. 126-134.

Calheiros, R.; Storch, M.; Alexandre, E; De Rose, C; e Breda, M. "Applying Virtualization and System Management in a Cluster to Implement an Automated Emulation Testbed for Grid Applications". 20th International Symposium on Computer Architecture and High Performance Computing. SBAC-PAD 2008, 29 de outubro a 1 de novembro 2008, Campo Grande, Brasil.

Charão, A; Santos, R. "Análise Comparativa de Desempenho do Hipervisor Xen: paravirtualização versus virtualização total". WSCAD 2008, 29 de outubro a 1 de novembro 2008, Campo Grande, Brasil.

EMA, Entreprise Management Solutions. (http://www.emscorporation.com). Acesso março de 2008.

Emeneker, W.; Jackson, D.; Butikofer, J.; Stanzione, D.; "Dynamic virtual clustering with Xen and Moab". Lecture Notes in Computer Sciences: International Symposium on Parallel and Distributed Processing and Application (ISPA). Workshops. Italy. pp. 440-451. 2006.

Globus. Globus Toolkit. Disponível em (http://www.globus.org). Acesso em out. 2008.

Goldberg, R.P. "Architecture of Virtual Machines". In Proceedings of Workshop on Virtual Computer. Systems.Cambridge, MA, USA. pp 74-112, 1973.

Goldberg, R.P. "Survey of virtual machine research". IEEE Computer, pp. 34-35, june 1974.

Henessy, J.L.; Patterson, D.A. "Computer Architecture: A quantitative approach". 4th edition. Morgan Kauffman Publishers, 2007.

IBM Corporation. "IBM Systems Virtualization Version 2 release 1 (2005)". Disponível em http://public.boulder,ibm.com/infocenter/eserver/v1r2/topic/eicay/eicay.pdf, acesso outubro 2008.

Jain, R. "Art of Computer Systems Performance Analysis". John Willey. 1991.

Keahey, K.; Foster, I.; Freeman, T.; Zhang, X.; "Virtual Workspaces: Achieving quality of service and quality of life on the *grid*". Scientific Programming, 13 (4):256-276, 2005.

Kozuck, H.; Satyanarayanan, M.; "Internet Suspend/Resume". Proc. 4th IEEE Workshop on Mobile Systems and Applications. Junho. 2002.

LinuxServer (http://linux-vserver.org). Acesso em março de 2008.

Microsoft. "Microsoft Virtual PC" (http://www.microsoft.com/Windows/virtualpc), acesso novembro 2007.

Microsoft. "Microsoft Virtual *Server*"

Neiger, G.; Santoni, A. et alli "Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization", Intel Technology Journal, v. 10, pp. 166-179, 2006.

Oliveira, R..; Carissimi, A.; Toscani, A.; *Sistemas Operacionais*. Editora Sagra-Luzzato, 3ª edição, 2004.

OSCAR. Open Source *Cluster* Application Resources. Disponível em (http://svn.oscar.openclustergroup.org/trac/oscar/wiki), acesso outubro 2008.

Popek, G.; Goldberg, R. "Formal requirements for virtualizable 3rd generation architectures". Communications of the ACM, v.17, n.7, pp. 412-421, 1974.

Righi, R.R.; Pilla, L.; Carissimi, A.; Navaux, P.O.A. "Controlling Process Reassignment in BSP Applications". 20th International Symposium on Computer Architecture and High Performance Computing. SBAC-PAD 2008, 29 de outubro a 1 de novembro 2008, Campo Grande, Brasil.

Robin, J.S.; Irvine, C.E. "Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor". Proc. 9th USENIX Security Simposium, 2000.

Rose, R. "Survey of Virtualization Techniques". (http://www.robertwrose.com). Acesso em fevereiro 2008.

Rosemblum, M. "The reincarnation of virtual machines". Queue Focus – ACM Press, pp 34-40, 2004.

Sapuntzakis, C.P.; Chandra, R.; Pfaff, B.; Chow, J.; Lam, M.S; Rosemblum, M.; "Optimizing the Migration of Virtual Computers". Proc. 5th Symp. On Operating Systems Design and Implementation. Dezembro, 2002.

Silberchatz, A.; Galvin, P; *Sistemas Operacionais*. (1a edição). Campus, Rio de Janeiro, 2001.

Singh, A."An introduction to virtualization"(http://www.kernelthread.com/publications) Acesso fevereiro 2008.

Smith, J.E, Nair, R. "The architecture of virtual machines". IEEE Computer, v.38, n.5, pp. 32-38, 2005.

Uhgli, R.; Neiger, G. et alli "Intel Virtualization Technology". Computer Journal, v.38, pp. 48-56. 2005.

Ung, D.; Cifuentes, C. Machine-adaptable dynamic binary translation. ACM SIGPLAN Notices, ACM Press New York, NY, USA, vol. 35, 7, july 2000, pp 41-51.

Vallée, G.; Naughton, T.; Scott, S.L.; "System management software for virtual environments". Proceedings of ACM International Conference on Computing Frontiers, Ischia, Italy, 2007.

Veiga, M.;Righi, R.R.; Maillard, N.; Navaux, P.O.A: Impacto da Migração de Máquinas Virtuais de Xen na Execução de Programas MPI. In VIII Workshop emSistemas Computacionais de Alto Desempenho. pp. 45-52. 2007

Virtual Workspace Project "Virtual Workspace Project" (http://workspace.globus.org). Acesso outubro 2008.

Virtuoso, Northwestern University. "Virtuoso: resource management and prediction for distributed computing using virtual machines". Disponível em http://www.csm.ornl.gov/harness . Acessado em outubro 2008.

VMWare (2008a) "A Performance Comparison of Hypervisors" (http://www.vmware.-com). Acesso janeiro 2008.

VMWare (2008b) "VMWare virtual networking concepts" (http://www.vmware.com) Acesso fevereiro 2008.

VMWare (2008c), VMWare (http://www.vmware.com)

Wikipedia. "Comparaison of virtual machines" Wikipedia, The Free Encyclopedia, Acesso novembro 2007.

Xen Source (2008a), "A Performance Comparaison of Commercial Hypervisors" (http:/ blogs.xensources.com/rogerk/wp-content/uploads/2007/03/*hypervisor*_perfor-mance_comparaison_1_0_5_with_esx-data.pdf). Acesso fevereiro 2008.

Xen Source (2008b), Xen Source (http://www.xensource.com). Acesso fevereiro 2008.