UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"
Pós-Graduação em Ciência da Computação

Rafael Augusto Teixeira

SW-V: Modelo de streaming de software baseado em técnicas de virtualização e transporte peer-to-peer

UNESP

2010

RAFAEL AUGUSTO TEIXEIRA

SW-V: Modelo de streaming de software baseado em técnicas de virtualização e transporte peer-to-peer

Dissertação apresentada para obtenção do título de Mestre em Ciência da Computação, área de Arquitetura de Computadores e Sistemas Distribuídos junto ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

Orientador: Marcos Antônio Cavenaghi

São José do Rio Preto, 26 de Março de 2010

Teixeira, Rafael Augusto.
SW-V: modelo de streaming de software baseado em técnicas de virtualização e transporte peer-to-peer / Rafael Augusto Teixeira. - São José do Rio Preto : [s.n.], 2010.
90 f. : il. ; 30 cm.

Orientador: Marcos Antônio Cavenaghi
Dissertação (mestrado) - Universidade Estadual Paulista, Instituto de Biociências, Letras e Ciências Exatas

1. Sistemas de computação virtual. 2. Sistemas operacionais. 3. Sistemas de processamento distribuído. I. Cavenaghi, Marcos Antônio. II. Universidade Estadual Paulista, Instituto de Biociências, Letras e Ciências Exatas. III. Título.

CDU – 004.75

Ficha catalográfica elaborada pela Biblioteca do IBILCE
Campus de São José do Rio Preto - UNESP

Agradecimentos

Em primeiro lugar, agradeço a Deus autor da maior obra científica que é a vida, todas as coisas foram criadas por Ele e sem Ele nada existiria. Agradeço por todas as vezes que iluminou minha mente e me mostrou o caminho a ser seguido, tanto na vida acadêmica quanto na vida cotidiana, pois tudo o que tenho e sou foi possível por Ele.

Agradeço também ao meu orientador Prof. Dr. Marcos Antônio Cavenaghi pela confiança em mim depositada desde o momento que me aceitou como orientando, pelo acompanhamento que tive durante o trabalho, pelas correções, e pelas idéias que foram agregadas ao trabalho.

Aos demais professores do programa de mestrado que com grande dedicação contribuíram para a formação das idéias deste trabalho, em especial a Prof. Dra. Roberta Spolon que acompanhou o desenvolvimento deste trabalho, ajudando sempre na organização das idéias, nas correções e submissões de artigos.

Agradeço a MSTech por entender a importância do programa de mestrado e me conceder a oportunidade de conciliar os estudos e o trabalho.

Aos meus amados pais Rosângela e José, que me proporcionaram desde minha infância condições para que eu nunca interrompesse os estudos, me incentivando sempre a buscar mais e nunca me acomodar. Agradeço ao carinho e amor que nunca faltou, que muitas vezes serviram de incentivo para a conclusão deste trabalho.

A toda minha família que de uma forma geral foram uma base para que este trabalho se concretizasse.

E por fim, agradeço a Karina, que passou de noiva a minha amada esposa durante o desenvolvimento deste trabalho, agradeço pela paciência principalmente nas horas que tive que dividir o tempo com o estudo. Agradeço pelo carinho, cuidado e amor que foram fundamentais para que esse trabalho fosse possível. Que possamos juntos, a cada dia, escrever página a página o nosso livro que nunca terá fim.

Resumo

Diante da constante evolução dos sistemas computacionais e seu potencial de processamento, a cada dia se torna necessário novas técnicas de aproveitamento destes recursos. Soluções que visam facilitar o gerenciamento da grande massa crescente de computadores e aplicações já se tornaram necessidades reais, não só soluções focadas nas grandes corporações, mas a cada dia, novas pequenas e médias empresas e outras organizações, como por exemplo escolas, fazem parte das estatísticas do uso de computadores e aplicações. Neste contexto, focando uma ferramenta compatível neste cenário de crescimento, este trabalho desenvolve um modelo de streaming de software chamado SW-V que é baseado no estudo de técnicas de virtualização e redes peer-to-peer que elimina a necessidade da instalação e configuração do software máquina-a-máquina para que ele seja utilizado. As características principais deste modelo que o destaca das soluções atuais são: um mecanismo de seqüenciamento dos recursos que são utilizados pelo software; um mecanismo de distribuição da imagem que utiliza a própria infra-estrutura de rede eliminando a necessidade de altos investimentos em servidores; um *middleware* de virtualização que não requer privilégios administrativos ou de sistema para ser executado; um mecanismo peer-to-peer assíncrono usado para aumentar a velocidade com que a distribuição da imagem do software ocorre para os clientes; e ainda possui suporte para que o usuário utilize as aplicações mesmo estando desconectado da rede. Durante o trabalho foram desenvolvidos pequenos módulos que vieram a comprovar que o modelo proposto pode ser adotado como base para a construção de uma solução robusta e escalável de virtualização de aplicação.

Abstract

In face of constant evolution of computing systems and the growing processing potential of computing, new techniques of use of these resources are necessary. Software solutions which intend make easy the management of this giant mass of computers and application are actually necessary, not only solution focused in big companies or large corporation, but each day, new small enterprises other segments like the educational organization became part of statistical of growing use of computer and software. In this scenario, this work formulate a model of software streaming based on study of virtualization techniques and peer-to-peer networks which eliminates the need of installation and configuration of determined software on each computer. The main characteristics of this model involve: a mechanism of sequencing of resources used by the software; a distribution mechanism of the software image using the available network infrastructure eliminating the need of high costs of investments and licensing; a peer-to-peer asynchronous distribution used to improve the sending of software image to the client computers; and still have off-line support where the user can execute de software even the computer is out of network and without server connection. Also some modules of mechanism proposed were developed to prove that all processes will works and all modules together can be used to build a robust and scalable framework of application virtualization.

Sumário

Lista de Abreviaturas e Siglas

ABI *Application Binary Interface*

API *Application Programming Interface*

ARM *Advanced RISC Machine*

BIOS *Basic Input/Output System*

CPU *Central Processing Unit*

DMA *Direct Memory Access*

GPL *General Public License*

HLL *High level Language*

HLL *High Level Language*

IBM *International Business Machine*

ISA *Instruction Set Architecture*

JIT *Just-in-time*

JVM *Java Virtual Machine*

MAC *Medium Access Control*

MMU *Memory Management Unit*

MV *Máquina Virtual*

PTE *Page Table Entry*

SO *Sistema Operacional*

TI *Tecnologia da Informação*

TLB *Translation Look aside Buffer*

VM *Virtual Machine*

VMM *Virtual Machine Monitor*

VMM *Virtual Machine Monitor*

VT-x *Virtualization Technology, x86*

App-V *– Solução de virtualização da Microsoft*

Lista de Figuras

Lista de Tabelas

Capítulo
1

Introdução

1.1 Contextualização

Os computadores quando foram inventados, eram gigantescos sistemas caros e difíceis de operar (Rose, 2004). Devido a este custo, se tornaram sistemas que compartilhavam tempo de processamento entre aplicações e usuários, sendo restrito apenas as empresas que tinham um poder aquisitivo maior. Com os avanços nas tecnologias de circuitos eletrônicos, o desenvolvimento dos microcomputadores se tornou cada vez mais aprimorado e comum. Tais computadores já possuíam preços inferiores aos preços dos computadores de grande porte que até então eram utilizados, popularizado o uso dos computadores por diversas empresas de pequeno e médio porte, antes prejudicadas pelo custo dos equipamentos.

Inversamente proporcional à queda do custo dos equipamentos, nos últimos anos a capacidade dos computadores processarem mais informações tem aumentado significativamente. Computadores modernos possuem capacidade suficiente para processar não somente o seu sistema operacional, mas também hospedar outras máquinas virtuais (VMs) executando cada uma, uma instância de um sistema operacional diferente (Barham *et al*, 2003), servindo de ponto de centralização para o gerenciamento de aplicações para os demais computadores na rede.

Não somente o poder computacional tem crescido exponencialmente na última década, mas também a necessidade das empresas de adaptarem seus sistemas a constantes mudanças organizacionais tem acompanhado esse avanço. De fato, cada vez mais empresas precisam de diversos serviços para garantir sua segurança, conectividade e aplicações que auxiliam na tomada de decisões. Levando em conta que muitas empresas e centros de dados utilizam somente de 10% a 15% do seu poder computacional (Golden & Scheffy, 2008), cada vez mais

a virtualização tem saído dos grandes centros de pesquisas e *DataCenters* para ocupar espaço de organizações de pequeno e médio porte de forma a maximizar o uso dos computadores.

Numa visão simplista, a virtualização nos permite que virtualmente e com um custo efetivamente menor tenha dois ou mais "computadores", executando cada um, um sistema operacional diferente, utilizando uma única peça de hardware (Golden & Scheffy, 2008). Além de permitir que muitas instâncias de sistemas operacionais usem o mesmo computador, a virtualização permite que o *hardware* seja padronizado na visão das aplicações, isolando a diversidade do hardware da camada de aplicação. Além desta característica (facilidade de migrar a VM para diferente *hardware* sem a necessidade de customização), o uso otimizado do espaço de armazenamento dos *DataCenters* tem sido atingido através da virtualização. Além da virtualização completa de um sistema operacional, as técnicas de virtualização combinadas com técnicas de distribuição eficientes de arquivos em rede podem proporcionar o streaming de software (através da virtualização de aplicação) permitindo que uma aplicação configurada apenas uma vez seja "replicada" para todos os computadores participantes da rede.

1.2 Motivações

Atualmente, virtualização é uma das técnicas mais empregadas e discutidas quando se fala em infra-estrutura de TI (Intel, 2008). O estudo na área da virtualização tem sido motivado pela crescente demanda por inovações, principalmente para atingir as novas tendências de múltiplos núcleos dos computadores. Aplicando as técnicas da virtualização, as empresas podem rapidamente atingir a capacidade máxima de processamento de seus computadores.

Outra forte motivação, como citado anteriormente, é que cada vez mais a virtualização não está se restringido aos *DataCenters* mas também está atingido usuários finais e pequenas organizações. Desde que os processadores *multi-core* se massificaram nos computadores pessoais, o poder de processamento destes computadores tem sido semelhante aos dos supercomputadores dos anos anteriores. Novas formas de se utilizar esse poder de processamento ocioso têm sido buscadas, e a virtualização tem se mostrado como uma ferramenta importante para atingir esse objetivo.

Atualmente, muitos dispositivos de diferentes hardware têm sido utilizados. Quanto uma peça de hardware ou o sistema operacional é reinstalado, todas as aplicações precisam ser novamente reinstaladas e reconfiguradas. A virtualização, em específico a de aplicação, pode facilmente abstrair a troca do hardware ou sistema operacional trazendo a portabilidade das aplicações e permitindo que sejam executadas sem serem instaladas de fato no computador.

Com a utilização em massa de computadores em todos os tipos de organizações, tem-se aumentado proporcionalmente a utilização de inúmeros softwares de gerenciamento e controle da organização em todos os seus níveis. Com isso a virtualização de aplicações tem se mostrado cada vez mais promissora como solução para redução de custos e dinamização do suporte a estes sistemas. O streaming de software utiliza ambientes de execuções isolados através de técnicas de virtualização. Isso abstrai a lógica da aplicação do sistema operacional em questão e do dispositivo físico, permitindo que uma única instalação seja distribuída ou enviada para diversos outros clientes, diminuindo os esforços e os custos de suporte na distribuição e configuração máquina-a-máquina do software corporativo.

De fato, um modelo bem estruturado de streaming de software que permita a execução de forma virtualizada pode reduzir os custos de infra-estrutura e manutenção do software de uma empresa. Além dos gastos de suporte, estas soluções eliminam problemas comuns como a migração de sistemas operacionais, uma vez que é necessária a reinstalação dos softwares quando há este tipo de migração. Uma vez configurado no novo SO, a aplicação poderá ser replicada para os demais computadores.

A demanda por sistemas de streaming de software não vem apenas dos ambientes corporativos. O software como um serviço (software-as-service) tem se tornado realidade. A virtualização de aplicação permite que mecanismos sejam criados para o controle do software como um serviço e não como uma instalação. Cada vez mais as aplicações estão passando de software *standalone* para aplicações distribuídas ou oferecidas via serviços on-line, acessadas de qualquer lugar por qualquer dispositivo via internet, eliminando o conceito antigo de a aplicação ter de ficar instalada somente no computador do usuário. A tendência futura da computação em nuvem tem conquistado mais espaço a cada dia. Com isso o software passará a ser um serviço e todo o software legado, caso não haja um modelo eficaz de streaming de

software, precisará ser adaptado. Assim, com certeza, milhares de software existentes hoje se tornariam incompatíveis com o novo modelo.

Outro ambiente que sugere uma demanda real por virtualização de aplicação é o ambiente educacional. Devido aos custos de manutenção dos computadores utilizados por alunos, várias escolas ou mesmo secretarias inteiras de educação dos estados brasileiros, tem adotado o modelo virtualizado de boot remoto, onde o disco das estações é protegido e virtualizado via rede no servidor. Após o uso do aluno, ao desligar o computador, o mesmo volta às configurações iniciais. Isto dá liberdade ao aluno de utilizar por completo o computador sem que haja desconfiguração ou infecção por vírus. O problema que, ao mesmo tempo em que protege a máquina, também tira a liberdade de um professor de instalar um software educacional para os alunos, pois ao ser desligado toda a instalação e configuração do mesmo serão perdidas.

Um dos entraves da massificação do uso desta tecnologia tem sido o custo, tanto o de licenças como o para se instalar uma infra-estrutura que comporte uma solução integrada de virtualização de aplicação que, devido à arquitetura hoje usada por esses soluções, torna necessária a utilização de vários servidores dedicados de alto desempenho. A maioria dos computadores presente nas organizações de pequeno porte, inclusive organizações educacionais, não comportariam esse modelo sem a atualização de suas máquinas. Além disso, a dificuldade da configuração das soluções torna inviável a sua utilização em organizações que não possuem núcleos próprios de gestão de tecnologia de informática ou um núcleo de técnicos dedicados.

1.3 Objetivos

Este trabalho tem por objetivo aplicar técnicas de virtualização e de redes peer-to-peer elaborando um modelo de streaming de software aliado a virtualização de aplicação, denominado de SW-V, capaz de se adequar em qualquer ambiente, requerendo menor investimento em hardware e reduzindo custos totais de suporte, tornando assim viável a utilização da virtualização de aplicações em ambientes de pequenas e médias empresas e outras organizações como o de escolas.

O modelo proposto utiliza-se de virtualização traduzindo as operações e chamadas de API da aplicação para o sistema operacional em que está sendo hospedado. Técnicas de

compartilhamento peer-to-peer deverão ser empregadas para dinamizar a distribuição das aplicações aos clientes na rede, visto que um dos principais problemas encontrados em trabalhos relacionados é o *overhead* do servidor por esses assumirem um papel de coprocessamento. No modelo proposto os usuários das aplicações devem também conseguir utilizá-las completamente mesmo que fora da rede a qual estava inserida.

Outro objetivo é que o modelo possa ser adotado no contexto da internet. As soluções existentes no mercado não se mostram eficientes e adequadas para serem usadas na internet por utilizarem uma forma de distribuição centralizada e em sua grande maioria não otimizada. Geralmente não dividem a imagem da aplicação e precisam que toda a imagem seja transferida para o cliente para que a aplicação comece a ser executada.

O objetivo deste trabalho não é mostrar o desenvolvimento de uma solução pronta. Seguindo o modelo proposto, alguns módulos e pequenos processos foram desenvolvidos a fim de servir de suporte para as afirmações na elaboração do modelo, e ao término deste trabalho será possível definir um fluxo ideal que uma solução de streaming de software com virtualização de aplicação deve ter, desde a montagem da imagem da aplicação com suas peculiaridades, até a sua execução virtualizada, deixando assim um modelo completo para desenvolvimento em trabalhos futuros.

1.4 Estrutura da Monografia

Este trabalho está organizado em 7 capítulos mais as referências bibliográficas utilizadas. Neste primeiro capítulo foi feito uma introdução ao assunto a ser tratado, apresentando o contexto ao qual se insere esse trabalho de pesquisa, as motivações do trabalho nesta área e os objetivos que este trabalho espera alcançar.

O capítulo 2 tem por objetivo apresentar a evolução da virtualização e como foram inicialmente concebidas. Além disso, mostrar como a virtualização passou do contexto dos *mainframes* e acabou seguindo para a arquitetura x86. Apresenta também as máquinas virtuais, com suas classificações e tipos.

No capítulo 3, serão abordadas as técnicas de virtualização, passando pelos desafios que a arquitetura x86 apresenta para a virtualização, a virtualização clássica e demais técnicas. Neste capítulo serão apresentados também exemplos de máquinas virtuais.

O capítulo 4 discute o conceito e a classificação da virtualização de aplicação, apresentando também exemplos de soluções já existentes com suas características.

O capítulo 5 apresenta de forma rápida e objetiva os conceitos envolvidos nas redes peer-to-peer, apresentando sua definição e tipos de arquiteturas.

O capítulo 6 mostra a arquitetura do modelo SW-V de streaming de software baseado em técnicas de virtualização. Neste capítulo serão expostos todos os módulos envolvidos no modelo proposto bem como suas principais características e como eles devem ser estruturados.

O capítulo 7 apresenta o protótipo desenvolvido para aplicar os conceitos do modelo proposto, mostra também a infraestrutura dos testes realizados com o protótipo.

O capítulo 8 é o capítulo conclusivo deste trabalho, mostra como os objetivos do trabalho foram alcançados. Ainda expõe as limitações atuais do modelo e faz propostas para trabalhos futuros que podem colaborar para o modelo.

E por fim, encerrando este trabalho, são apresentadas as fontes bibliográficas utilizadas.

Capítulo
2

Virtualização

Desde a criação dos computadores eletrônicos por Von Neumann na década de 40, até os dias atuais, os sistemas computacionais vêm sofrendo constante evolução. Até o início da década de 80, os computadores eram grandes e caros. Segundo a lei de Grosch, o poder computacional de uma CPU era proporcional ao quadrado de seu preço (EIN-DOR, 1985). Desta forma, as empresas tinham um elevado custo com enormes máquinas para terem seus requisitos computacionais atendidos.

A partir da segunda metade da década de 80, o desenvolvimento de microprocessadores e avanços nas tecnologias de redes de computadores mudou esta situação e possibilitaram o surgimento de técnicas de compartilhamento de recursos das máquinas.

Virtualização é um conceito primeiramente desenvolvido na década de 60 para compartilhar *hardware* de grandes *mainframes*. Hoje, computadores baseados na arquitetura x86 estão frentes ao mesmo problema do sub-aproveitamento de recursos que os *mainframes* encontraram naquela época. Várias empresas como VMWare® e Microsoft® a partir da década de 90 visualizaram essa oportunidade e entraram no mercado desta arquitetura (VMWare Inc, 2008).

2.1 No início dos *mainframes*

Em junho de 1959, Christopher Strachey publicou um artigo com o titulo *Time Sharing in Large Fast Computers* na conferencia internacional de processamento de informações, em Nova York (Singh, 2008). Anos mais tarde, em 1974, ele mesmo admitiu que na época do artigo (anos 50) o conceito de *Time Sharing* era confuso e mal entendido, e que seu artigo tratava-se sobre técnicas de multi-programação para evitar que os programas dos supercomputadores esperassem respostas dos periféricos lentos.

O conceito de multi-programação para *spooling* de aplicações pôde ser observado no começo da década de 60 com o projeto ATLAS. Este projeto foi um esforço entre a universidade de Manchester e a companhia Ferranti Ltda. De acordo com os projetistas, o ATLAS era formado por um supervisor de rotinas, que era ativado por interrupções ou por instruções nos programas em execução. Uma espécie de precursor da "máquina virtual" como conhecemos hoje foi utilizada para o supervisor e outra usada para executar os programas.

Na metade da década de 60, o conceito de máquina virtual foi utilizado pela IBM® como uma forma de dividir recursos dos *mainframes* através de máquinas virtuais separadas de forma lógica. Essas partições permitiam aos *mainframes* trabalharem de forma multitarefa, ou seja, executando múltiplas aplicações e processos ao mesmo tempo. O projeto M44/44X teve sua arquitetura baseada em máquinas virtuais, a máquina principal foi a IBM7044 (corresponde a M44 do nome do projeto) e as máquinas adjacente, uma cópia da principal, operando no mesmo espaço de memória do M44, implementado através de memória virtual e multi-programação.

Nesta época a IBM proveu um IBM 704, uma série de atualizações de hardware e acesso para alguns de seus engenheiros de sistema para o MIT (*Massachusetts Institute of Technology*). Sobre as máquinas da IBM foi que o MIT desenvolveu o CTSS (*Compatible Time Sharing System*), onde o programa supervisor agora tratava I/O dos consoles, agendamento das tarefas, armazenamento temporário e recuperação de programas durante o *swap*, monitoramento de discos e etc. O supervisor tinha controle total sobre todas as interrupções das máquinas virtuais em execução. Durante essa época a IBM trabalhava na montagem da família de computadores 360. O MIT no final de 1963 desenvolveria o projeto MAC baseado nos avanços em *time sharing* do CTSS, que mais tarde originaria o Multics. Multics (Multiplexed Information and Computing Service), o qual era um conceito muito adiante do seu tempo, foi o primeiro sistema operacional de tempo compartilhado. A última instalação operacional do Multics foi desligada apenas em 31 de outubro do ano de 2000. Neste projeto o MIT foi apoiado pela *General Electrics* usando o GE645 ao invés das máquinas da IBM.

Sem levar em consideração essa "perda", a IBM foi uma importante força no campo de pesquisa de virtualização. Um grande número de sistemas virtualizados foram inventados com base nas pesquisas executadas pela IBM. Tipicamente as máquinas virtuais da IBM eram

cópias idênticas do *hardware* base. Um componente chamado Virtual Machine Monitor (VMM), executava diretamente sobre o *hardware* real. Múltiplas máquinas virtuais podiam ser criadas através do VMM e cada uma dessas instâncias executava seu próprio sistema operacional. A herança hoje deixada pelas máquinas virtuais da IBM é muito respeitada e considerada plataformas robustas de computação (Singh, 2008).

2.2 A necessidade de virtualização para a arquitetura x86

Durante a década de 80, a virtualização foi esquecida quando aplicações cliente-servidor, os ainda inexpressivos servidores na arquitetura x86 e os computadores desktop, estabeleceram um novo modelo de computação distribuída (VMWare Inc, 2008). Ao invés de compartilhar recursos centralizando no modelo dos *mainframes*, organizações usaram o baixo custo dos sistemas distribuídos para montar suas infra-estruturas. A larga adoção do Windows® e o emergente uso do Linux como sistema operacional servidor no final da década de 80 e inicio da década de 90, consolidaram o uso da arquitetura x86 em pequenas e médias empresas.

O crescente uso dessa arquitetura introduziu novos desafios operacionais e na infra-estrutura de TI, dentre eles:

- Baixa utilização da Infra-estrutura: Servidores x86 tipicamente atingiam somente de 10 a 15% do total de sua capacidade, de acordo com o IDC (*Internantional Data Corporation*). Tipicamente as organizações executavam apenas uma aplicação por servidor, para evitar o risco de uma aplicação afetar a disponibilidade da outra.

- Custo alto no crescimento da infra-estrutura: O custo para se aumentar a infra-estrutura começava aumentar rapidamente. O aumento implicava na estrutura física, resultando em consumo de energia, sistemas de refrigeração e etc. O custo era maior levando em consideração que o nível de complexidade dos sistemas tornava-se cada vez mais carente de pessoas especializadas e experientes, elevando ainda mais o custo do gerenciamento de TI.

- Insuficiente recuperação de falhas e desastres: Organizações crescentemente afetadas por falhas do servidor, desastres naturais e ataques terroristas. Não existiam mecanismos suficientemente eficazes para estes problemas.

Devido ao constante uso da arquitetura x86 como servidor, a virtualização foi novamente colocada como estratégica, e necessária para resolver problemas desta nova infra-estrutura. A partir daí, rapidamente técnicas de virtualização foram aprimoradas e outras desenvolvidas.

2.3 Abstração e virtualização

Sistemas de computadores são projetados com basicamente três componentes principais: hardware, sistema operacional e aplicações, como mostra a figura 1:

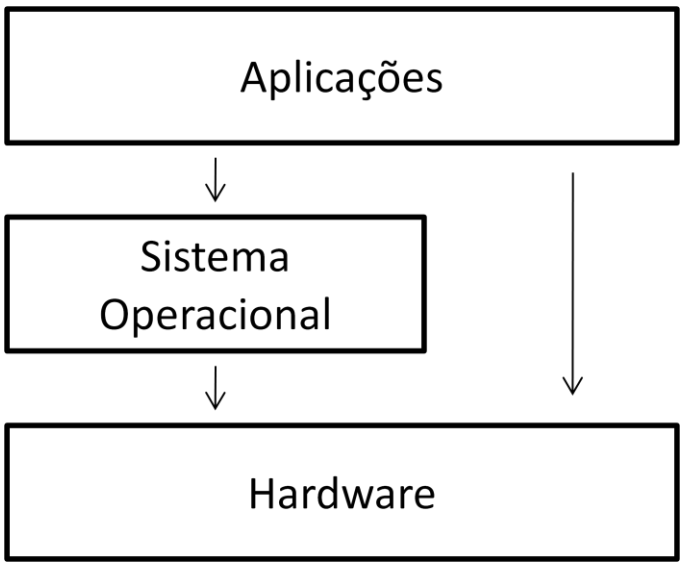


Figura 1 – Principais componentes de um sistema computacional (Laureano, 2006)

Estes sistemas são projetados em interfaces bem definidas que criam diferentes níveis de abstração (SMITH & NAIR, 2005) permitindo que diferentes equipes desenvolvam ao mesmo tempo *software* e *hardware*. O conjunto de instruções da arquitetura (ISA – *Instruction Set Architecture*) simplifica os detalhes de implementação de baixo nível e reduz a complexidade do processo de criação. Programadores podem criar seus programas sem saber

a organização física e lógica do *hardware* através da abstração. Por exemplo, Intel e AMD implementaram seus processadores sobre o conjunto de instruções Intel IA-32 (x86), enquanto os desenvolvedores da Microsoft desenvolveram o sistema operacional sobre esta especificação ISA, sendo, portanto compatível com ambos processadores.

Um exemplo típico dessa estrutura em níveis de abstração, separados por interfaces bem definidas, são os subsistemas de rede e de disco em um sistema operacional convencional. No sub-sistema de arquivos, cada nível abstrai um problema, como por exemplo, escalonamento de acesso ao disco e gerenciar *buffers* e *cachê*, como mostra a figura Figura 2:

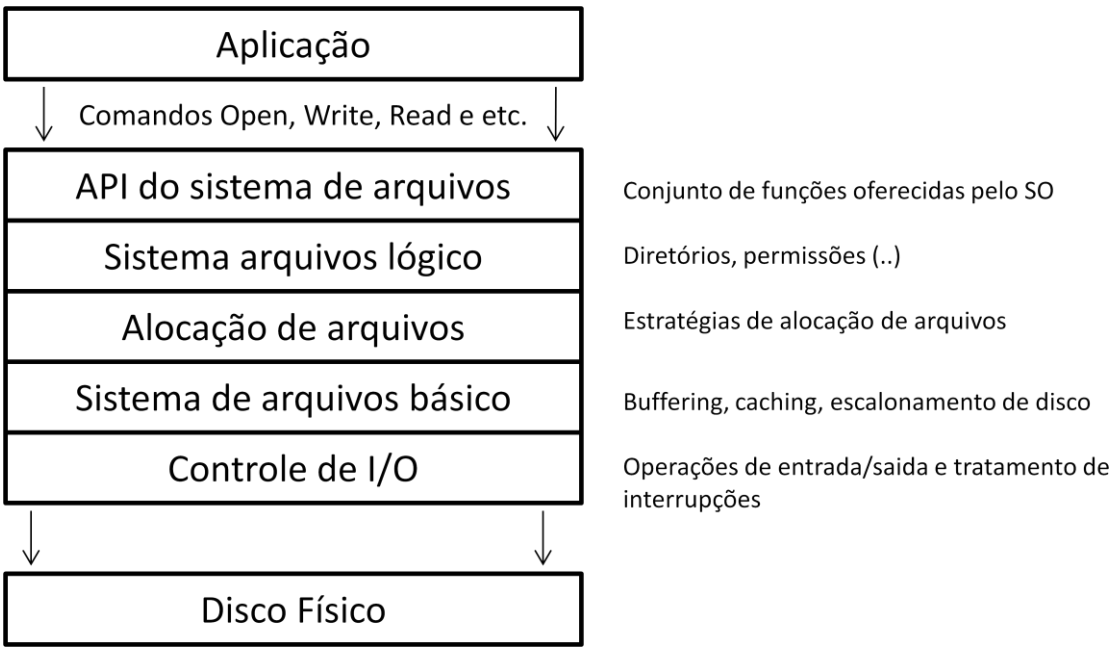


Figura 2 - Níveis de abstração em um subsistema de disco.

As interfaces existentes entre os diferentes componentes de um sistema computacional são:

- ***Conjunto de instruções* (*ISA – Instruction Set Architecture*)**: é a interface básica entre o hardware e o software, sendo constituída pelas instruções em código de máquina aceitas pelo processador e todas as operações de acesso aos recursos do hardware (acesso físico à memória, às portas de I/O, etc.). Essa interface é sub-dividida em duas partes:

- ***Instruções de usuário* (*User ISA*)**: compreende as instruções do processador e demais itens de hardware acessíveis aos programas do usuário, que executam com o processador operando em modo não-privilegiado;

- ***Instruções de sistema* (*System ISA*)**: compreende as instruções do processador e demais itens de hardware, unicamente acessíveis ao núcleo do sistema operacional, que executa em modo privilegiado;

- ***Chamadas de sistema* (*syscalls*)**: é o conjunto de operações oferecidas pelo núcleo do sistema operacional aos processos dos usuários. Essas chamadas permitem um acesso controlado das aplicações aos dispositivos periféricos, à memória e às instruções privilegiadas do processador.

- ***Chamadas de bibliotecas* (*libcalls*)**: bibliotecas oferecem um grande número de funções para simplificar a construção de programas; além disso, muitas chamadas de biblioteca encapsulam chamadas do sistema operacional, para tornar seu uso mais simples. Cada biblioteca possui uma interface própria, denominada ***Interface de Programação de Aplicações* (*API – Application Programming Interface*)**. Exemplos típicos de bibliotecas são a *LibC* do UNIX (que oferece funções como fopen e printf), a GTK+ (*Gimp ToolKit*, que permite a construção de interfaces gráficas) e a SDL (*Simple DirectMedia Layer*, para a manipulação de áudio e vídeo).

A figura 3 apresenta uma visão conceitual da arquitetura de um sistema computacional:

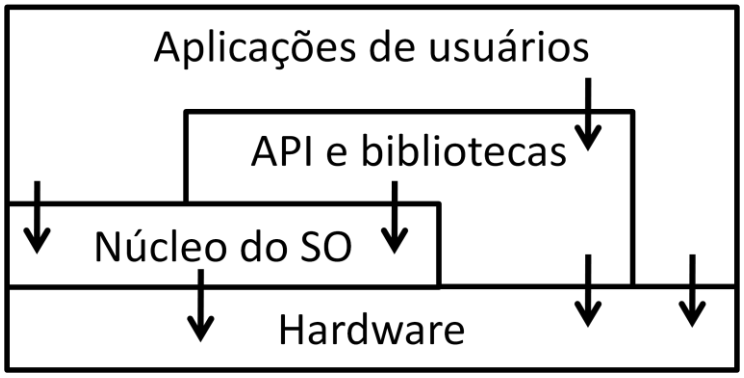


Figura 3 - Componentes e interfaces de um sistema computacional (Laureano, 2006)

Infelizmente, estas interfaces bem definidas também têm suas limitações. Subsistemas e componentes projetados para uma interface, não irão trabalhar com outra. Aplicações são distribuídas e compiladas respeitando o conjunto de instruções do processador em modo usuário (*User ISA*) e o conjunto de chamadas de sistemas oferecido pelo sistema operacional (*syscalls*), tendo a visão conjunta dessas duas interfaces (*User ISA + syscalls*) denominada de *Interface Binária de Aplicação* (*ABI - Application Binary Interface*). Da mesma maneira, um sistema operacional só pode ser executado respeitando a interface ISA (User/System) do hardware para o qual foi projetado. A figura 4 mostra as interfaces ISA e ABI (SMITH & NAIR, 2005):

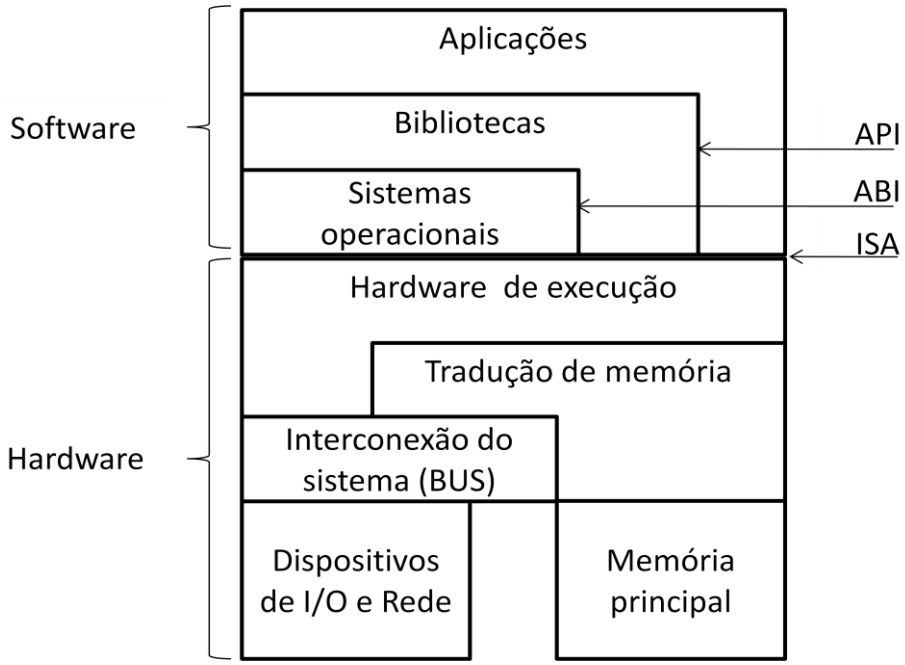


Figura 4 - Interfaces ISA e ABI (SMITH & NAIR, 2005)

A definição de interfaces, mesmo que útil, por facilitar o desenvolvimento, torna pouco flexível as interações entre eles: um sistema operacional só funciona sobre o hardware (ISA) a qual foi projetado e uma biblioteca só trabalha sobre a ABI estabelecida.

Virtualização provê um caminho para resolução desses problemas. Virtualizar um sistema ou componente do mesmo, como um processador, memória ou dispositivo de I/O, pode mapear as interfaces e os recursos de um sistema para outro. Diferente da abstração, a virtualização não ajuda necessariamente a simplificar ou esconder detalhes de implementação. A abstração provê uma interface simplificada para atingir os recursos, como um disco de

armazenamento, por exemplo. A virtualização provê uma interface diferente ou diferentes recursos no mesmo nível de abstração, proporcionando uma camada de compatibilidade entre a aplicação (ou sistema operacional inteiro) e o hardware ou subsistema hospedeiro.

2.4 Máquinas virtuais

Como discutido anteriormente, o conceito de máquina virtual ou simplesmente VM (sigla em inglês para *virtual machine*), foi concebido pela IBM na tentativa de compartilhar tempo e recursos dos caros *mainframes* da época (Rose, 2004). Na década de 60 e 70, a máquina virtual era definida como uma cópia perfeita e isolada de uma máquina real (Popek & Goldberg, 1974) (Goldberg & Mager, 1979) (Creasy, 1981).

O termo virtual foi descrito na década de 1960 como uma abstração de software para um sistema físico (máquina virtual) (Laureano, 2006). Com o passar dos anos o termo englobou um grande número de abstrações, como por exemplo, a máquina virtual Java (JVM), que na verdade não virtualiza um sistema completo real (Rosenblum, 2004), conforme veremos nas seções seguintes.

Como visto anteriormente, as interfaces padronizadas entre os componentes do sistema computacional permitem o desenvolvimento facilitado, mas também apresentam problemas de interoperabilidade, por exemplo, não é possível executar um programa compilado para um processador ARM num processador Intel, as instruções entre os dois processadores são diferentes, da mesma forma uma aplicação Windows não executará no Linux.

Para superar este problema é possível criar um ambiente virtual (camada de virtualização) construída em software. Este software será responsável por oferecer os serviços de uma determinada interface de uma forma com que seja compreendido pelo componente subseqüente.

Usando serviços oferecidos por uma determinada interface de sistema, a camada de virtualização constrói outra interface de mesmo nível, de acordo com as necessidades dos componentes de sistema que farão uso dela. A nova interface do sistema vista através dessa camada de virtualização é denominada *máquina virtual*, como mostra a figura 5.

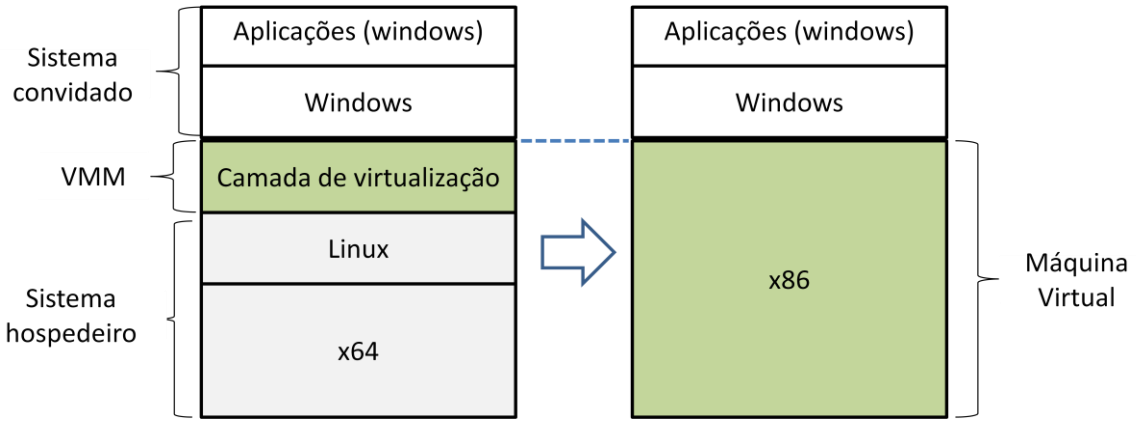


Figura 5 - Máquina virtual (Laureano, 2006)

Um ambiente virtualizado por uma máquina virtual pode ser dividido em três partes básicas: ***o sistema real***, nativo ou hospedeiro (*host system*) onde existe os recursos reais de *hardware* e *software* do sistema; o ***sistema virtual***, também pode ser chamado de sistema convidado (*guest system*), que é executado sobre o sistema computacional virtualizado, sendo que vários sistemas virtualizados podem coexistir, estando em execução simultânea no mesmo sistema real; e a ***camada de virtualização***, denominada *hipervisor* ou *monitor de máquina virtual (VMM – Virtual machine monitor)*, que é responsável por criar a compatibilidade entre as interfaces.

Existem várias técnicas de virtualização que serão discutidas na seção 3 deste documento, a saber:

- ***Virtualização completa***: um sistema convidado e suas aplicações são executadas diretamente sobre o hardware virtualizado para o qual foi desenvolvido, sendo que o sistema operacional não precisa ser modificado.

- ***Paravirtualização***: o sistema a ser virtualizado (*guest*) sofre modificações para aperfeiçoar a interação como *hipervisor*.

- ***Tradução dinâmica***: as instruções das aplicações são traduzidas durante a execução em instruções mais eficientes naquela plataforma.

- ***Assistência por hardware***: Onde algumas instruções que facilitam a virtualização estão sendo incorporadas diretamente no hardware.

2.5 Virtual Machine Monitor

A idéia de um monitor de máquinas virtuais (VMM) anda de mãos dadas com a virtualização (Rose, 2004). O VMM é o componente de software que hospeda o sistema convidado. De fato, o VMM é conhecido com o hospedeiro (*host*) e as máquinas virtualizadas como convidados (*guest*). O VMM é uma camada de software que abstrai a camada física de recursos para ser usada pelas máquinas virtualizadas (XenSource, 2007). O VMM provê um "processador virtual" e outras versões virtualizadas dos dispositivos do sistema como dispositivos de I/O, armazenamento, memória e etc. O VMM ainda proporciona um isolamento entre a máquina virtual e seu hospedeiro, como mostra a figura 6, para que falhas não venham a afetar uma a outra.

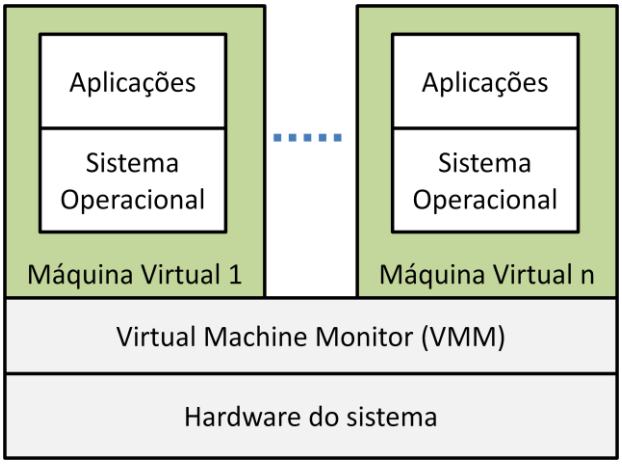


Figura 6 - VMM ou hipervisor: camada de virtualização (Adams & Agesen, 2006)

2.5.1 Propriedades do VMM

Para atender de forma eficiente o sistema hóspede, um *hipervisor* precisa atender a alguns requisitos básicos, que são: prover um ambiente de execução idêntico a uma máquina real no ponto de vista lógico; e ter controle completo sobre os recursos do sistema real (*host*). A partir daí Popek e Goldberg (Goldberg, 1973) (Popek & Goldberg, 1974) estabeleceram as seguintes propriedades de um *hipervisor* ideal:

- ***Equivalência***: prover um ambiente de execução idêntico ao da máquina real original. Todo programa executando em uma máquina virtual deve se comportar da mesma forma que o faria em uma máquina real;

- ***Controle de recursos***: possuir o controle completo dos recursos da máquina real: nenhum programa executando na máquina virtual deve ter acesso a recursos que não tenham sido explicitamente alocados a ele pelo *hipervisor*, que deve intermediar todos os acessos. Além disso, a qualquer instante o *hipervisor* pode resgatar recursos alocados.

- ***Eficiência***: grande parte das instruções do processador virtual (provido pelo *hipervisor*) deve ser executada diretamente pelo processador da máquina real. As instruções da máquina virtual que não puderem ser executadas pelo processador real devem ser interpretadas pelo *hipervisor* e traduzidas em ações equivalentes no processador real, caso contrário, devem ser executadas diretamente no hardware.

Além destas propriedades, outras características derivadas são associadas ao *hipervisor* (Rosenblum, 2004):

- **Isolamento**: garantir que um software em execução em uma máquina virtual não possa ter influencia ou modificar outro software em execução no *hipervisor* ou em outra máquina virtual. Garante que erros de software ou aplicações maliciosas possam ser contidos.

- ***Inspeção***: o *hipervisor* tem acesso e controle sobre todas as informações do estado interno da máquina virtual, como registradores do processador, conteúdo de memória, eventos etc.

- ***Gerenciabilidade***: como cada máquina virtual é uma entidade independente das demais, o *hipervisor* deve ter mecanismos para gerenciar o uso dos recursos existentes entre os sistemas convidados.

- ***Encapsulamento***: como o *hipervisor* tem acesso e controle sobre o estado interno de cada máquina virtual em execução, ele pode salvar *checkpoints* de uma máquina virtual, periodicamente ou em situações especiais (por exemplo, antes de uma atualização de sistema operacional). Esses *checkpoints* são úteis para retornar a máquina virtual a estados anteriores (*rollback*), para análises

post-mortem em caso de falhas, ou para permitir a migração da máquina virtual entre *hipervisors* executando em computadores distintos.

- ***Recursividade***: deve ser possível executar um *hipervisor* dentro de uma máquina virtual, produzindo um novo nível de máquinas virtuais, mostrado na figura 7. Neste caso, a máquina real é normalmente denominada *máquina de nível 0.*

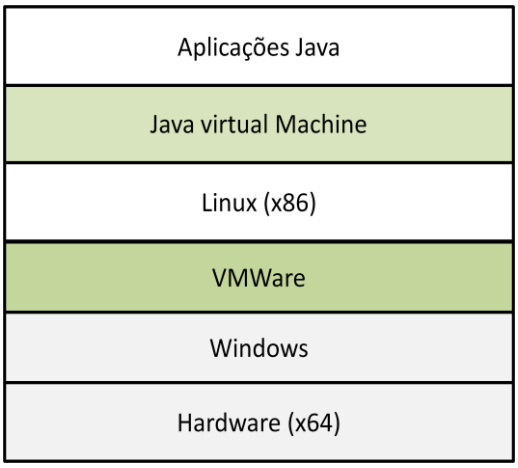


Figura 7 - Níveis de virtualização (Laureano, 2006)

2.5.2 Teorema de Popek e Goldberg

"Para qualquer computador convencional de terceira geração, um hipervisor pode ser construído se o conjunto de instruções sensíveis daquele computador for um sub-conjunto de seu conjunto de instruções privilegiadas." (Popek & Goldberg, 1974)

Para compreender melhor a afirmação definimos: *computador convencional de terceira geração* como sistema de computação que segue a arquitetura de Von Neumann (suporte memória virtual e dois modos de operação: modo usuário e modo privilegiado); *Instruções sensíveis* como sendo aquelas que podem consultar ou alterar o status do processador, ou seja, os registradores que armazenam o status atual da execução na máquina real; *Instruções privilegiadas* que são acessíveis somente por meio de códigos executando em nível privilegiado (código de núcleo). Caso um código não-privilegiado tente executar uma instrução privilegiada, uma exceção (interrupção) é gerada, ativando uma rotina de tratamento previamente especificada pelo núcleo do sistema real (Laureano, 2006).

Este teorema afirma que toda instrução sensível deve ser privilegiada. Quanto uma instrução sensível for executada por um programa não-privilegiado (núcleo do sistema convidado, por exemplo), provocará a ocorrência de uma interrupção. Isto pode ser usado para ativar uma rotina de interpretação no *hipervisor* que irá simular uma instrução sensível, ou seja, irá interpretar essa instrução. Sendo assim, quanto maior o número de instruções sensíveis, maior o volume de interpretação e menor o desempenho da máquina virtual.

2.6 Classificação das VMs

As máquinas virtuais podem ser divididas em duas grandes famílias (SMITH & NAIR, 2005):

- ***Máquinas virtuais de aplicação* (*Process Virtual Machines*)**: ambientes de máquinas virtuais destinados a suportar apenas processos ou aplicações convidadas específicas. A máquina virtual Java é um exemplo desse tipo de ambiente.

- ***Máquinas virtuais de sistema* (*System Virtual Machines*)**: ambientes de máquinas virtuais construídos para suportar sistemas operacionais convidados completos, com aplicações convidadas executando sobre eles. Exemplos: *VMware*®, *VirtualBox*®, *VirtualPC*® entre outros.

A figura 8 mostra estes dois tipos de ambientes:

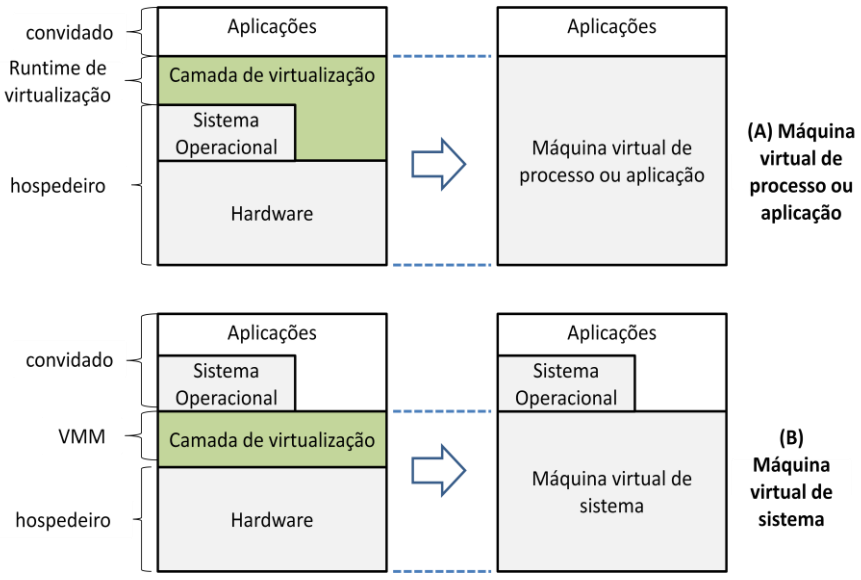


Figura 8 - Máquinas virtuais de aplicação (A) e de sistema (B) (SMITH & NAIR, 2005)

2.6 Máquina virtual de aplicação

Máquina virtual de aplicação ou de processo (Process Virtual Machine) provê uma ABI ou API (*Application Binary Interface e Application Programming Interface*, respectivamente*;* discutido na seção *Abstração e virtualização)* para aplicações usuário. Em suas várias implementações, uma VM de aplicação oferece replicação, emulação e otimização (SMITH & NAIR, 2005).

2.6.1 Sistemas Multi-programados

A VM de aplicação mais comum é tão ambígua que pouco é lembrada como uma VM. Muitos sistemas operacionais podem executar diversas aplicações e processos ao mesmo tempo através da multi-programação, que dá a ilusão ao processo de ter uma máquina completa para ele: cada processo tem seu próprio espaço de endereço, registradores, nível de isolamento e estrutura de arquivos. O sistema operacional gerencia e compartilha o tempo de uso dos recursos do *hardware*. De fato o sistema operacional provê em certo nível uma VM de aplicação para cada um dos processos executando simultaneamente.

2.6.2 Emuladores e tradução binária dinâmica

O desafio maior para a VM de processo é de suportar aplicações compiladas para um diferente conjunto de instruções em relação ao que o *host* executa. O caminho mais direto de realizar a emulação é através da *interpretação.* Um programa interpretador pega, decodifica e emula a execução de instruções individuais da aplicação *guest*. Isto pode ser relativamente um processo lento, exigindo dezenas de instruções do *host* para cada instrução fonte. Uma melhor desempenho pode ser obtida através da *tradução binária dinâmica* (*dynamic binary translation),* que converte instruções do *guest* para o *host* em blocos ao invés de traduzir instrução por instrução, e salva esses blocos em um *cache* em software para reutilizar caso uma instrução seja novamente chamada. Execução repetida das instruções traduzidas diminui relativamente o alto *overhead* da tradução. Ainda, para reduzir as perdas de desempenho os tradutores podem realizar otimizações nas instruções durante a tradução.

2.6.3 VMs de linguagem de alto-nível

Para VMs de processo, portabilidade de plataforma é um objetivo claro. Entretanto, emular uma arquitetura convencional em outra exige um estudo caso-a-caso e um esforço de programação. Uma portabilidade completa de plataforma é mais prontamente atingida projetando uma VM como parte do ambiente de desenvolvimento de uma aplicação de linguagem de alto-nível (*HLL – High level Language*). A VM HLL resultante não corresponde a uma plataforma real, ao contrário, é projetada para prover portabilidade.

A figura 9 mostra a diferença entre uma compilação numa plataforma convencional e um ambiente de VM HLL:

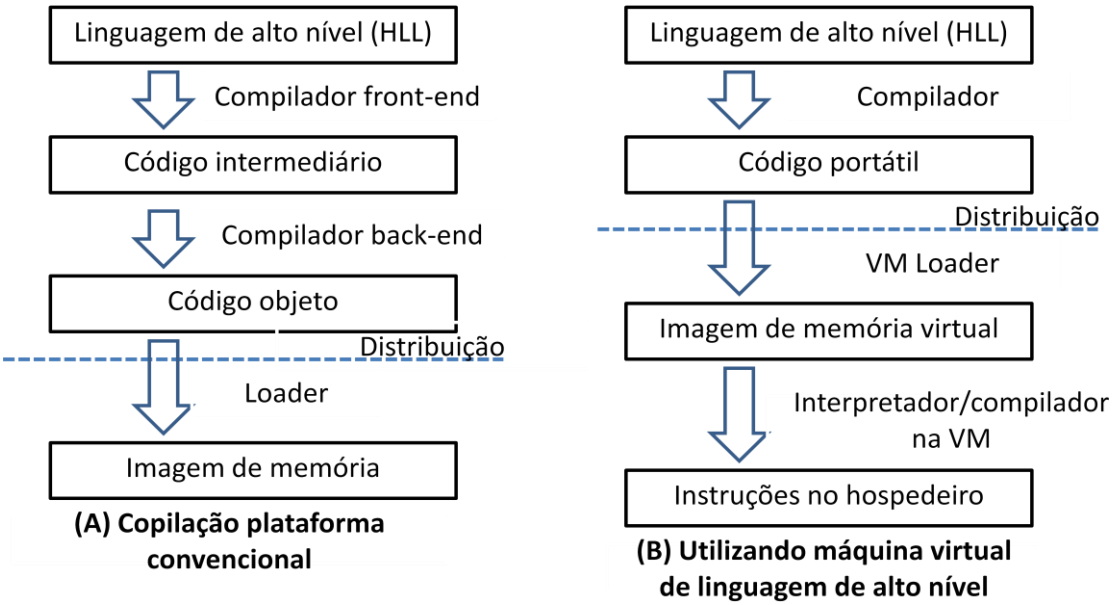


Figura 9 - VM de linguagem de alto-nível (SMITH & NAIR, 2005)

Num sistema convencional (a), um compilador *front-end* primeiramente gera um código intermediário que é similar ao código de máquina. Então um gerador de código *back-end* usa esse código intermediário para gerar um binário contendo códigos de máquina para uma específica ISA e sistema operacional. Este arquivo binário é distribuído e executado somente sobre a plataforma que possui a combinação ISA/OS correta.

Em um ambiente VM HLL (b), um compilador *front-end* gera um código de máquina abstrato em uma ISA virtual especificada pela interface da VM. Este código virtual, junto

com um metadado associado contendo informações sobre ele, é distribuído para ser executado em diferentes plataformas. Cada plataforma *host* implementa uma VM capaz de carregar e executar a ISA virtual e um conjunto de rotinas especificada por uma API padronizada. VM HLL mais sofisticadas recompilam o código virtual para máquina *host* para ser executado diretamente sobre a plataforma.

Uma vantagem da VM HLL é que a aplicação de software pode ser facilmente portada para outra plataforma, uma vez que a VM e bibliotecas são implementadas para várias plataformas. A arquitetura Sun Microsystems Java VM e a Microsoft .Net Framework, são exemplos de VM HLL largamente utilizadas em todo mundo.

2.7 Máquina virtual de sistema

Uma VM de sistema provê um completo ambiente em que um sistema operacional e muitos processos possivelmente pertencentes a múltiplos usuários podem coexistir (SMITH & NAIR, 2005). Usando uma VM de sistema, um único *hardware host* pode suportar múltiplos, isolados e protegidos sistemas operacionais simultaneamente. A VM de sistema emergiu durante a década de 60 e inicio da década de 70, e foi a origem do termo virtual machine conforme estudado anteriormente com o inicio dos *mainframes*.

Em uma VM de sistema, o *hipervisor* (VMM) primariamente provê a replicação de plataforma. O ponto central é a divisão de um conjunto de recursos de *hardware* entre múltiplos ambientes convidados cada um com seu sistema operacional. O *guest* (sistema operacional convidado) e suas aplicações são gerenciados de forma oculta pelo *hipervisor*. Quando um SO *guest* realiza uma instrução privilegiada ou operação que diretamente interage com o hardware compartilhado, o *hipervisor* intercepta a operação, checa por concorrência, e então realiza a operação requisitada pelo *guest*, sendo que ele (*guest)* não sabe deste processo.

As VMs de sistema podem ser divididas em dois tipos:

Modelo clássico: (ou ***nativo***) Da perspectiva do usuário, as VMs de sistema provêem essencialmente a mesma funcionalidade, mas difere em seus detalhes de implementação. A abordagem clássica coloca o *hipervisor* diretamente sobre o *hardware* e a VM no topo dessa estrutura (Goldberg, 1973) (Popek & Goldberg, 1974), sendo que o *hipervisor* é executado no modo mais privilegiado, enquanto todos os outros sistemas têm privilégios mais reduzidos.

Modo hospedado: Uma implementação alternativa coloca o sistema virtualizado sobre o topo de um sistema operacional hospedeiro (este passa a ser o *host*), passando então para uma VM hospedada. Uma vantagem desse modelo é que a VM é instalada como uma aplicação típica. O software virtualizado pode ainda utilizar recursos do sistema operacional *host* desde que este compartilhe tais recursos.

A figura 10 ilustra esses dois modelos de virtualização:

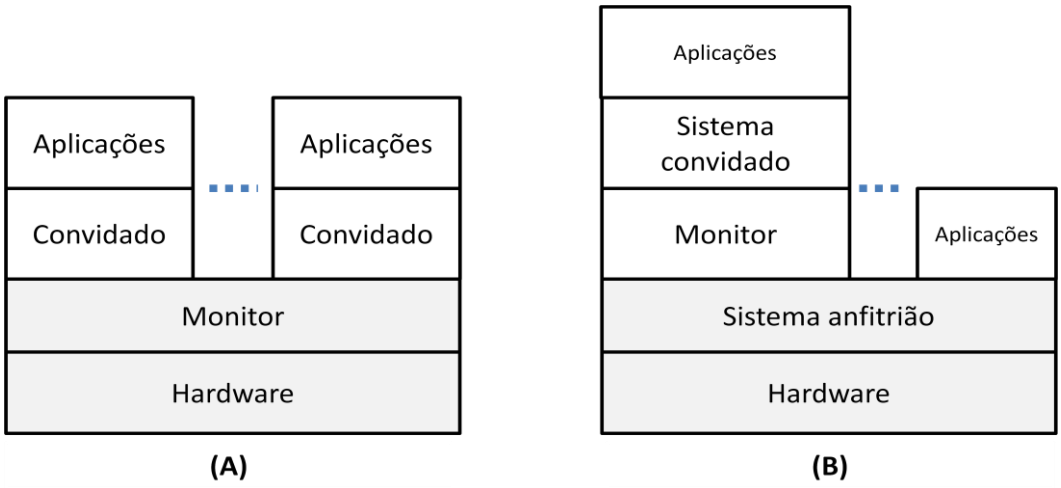


Figura 10 - VMs de sistema: (a) modelo clássico(ou nativo), (b) modelo hospedado

Estes dois tipos de VM de sistema, são raramente utilizados em sua forma conceitual em implementações reais (Laureano, 2006). Na prática abordagens ***híbridas*** são inseridas nas arquiteturas para aperfeiçoar e aumentar o desempenho do sistema, principalmente em relação as operações que envolvem dispositivos de I/O.

No modelo clássico (ou nativo) de VMs de sistema, o sistema convidado (guest) pode acessar diretamente o hardware. Essa forma de acesso é implementada no núcleo do *hipervisor*. Essa otimização ocorre, por exemplo, no subsistema de gerência de memória do Xen, como mostram as figuras 11 e 12.

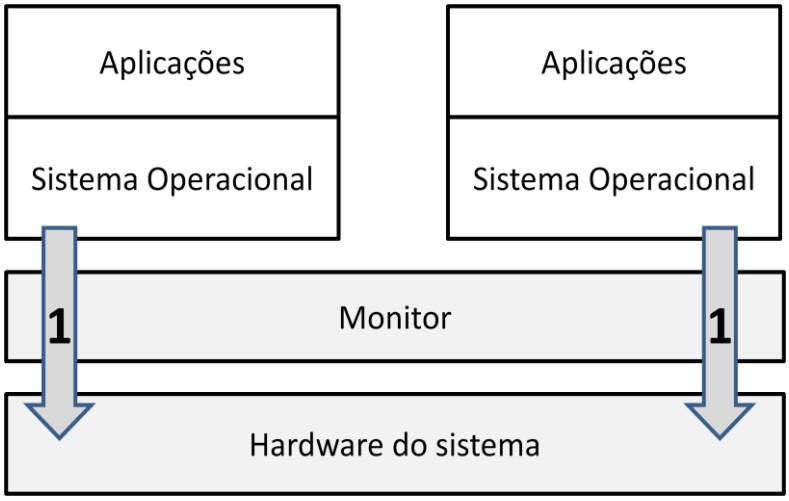


Figura 11 - Abordagem híbrida do modelo clássico da VM de sistema: (1) o sistema guest pode acessar diretamente o hardware. (Laureano, 2006)

No modelo Hospedado de VM de sistema as otimizações podem ser:

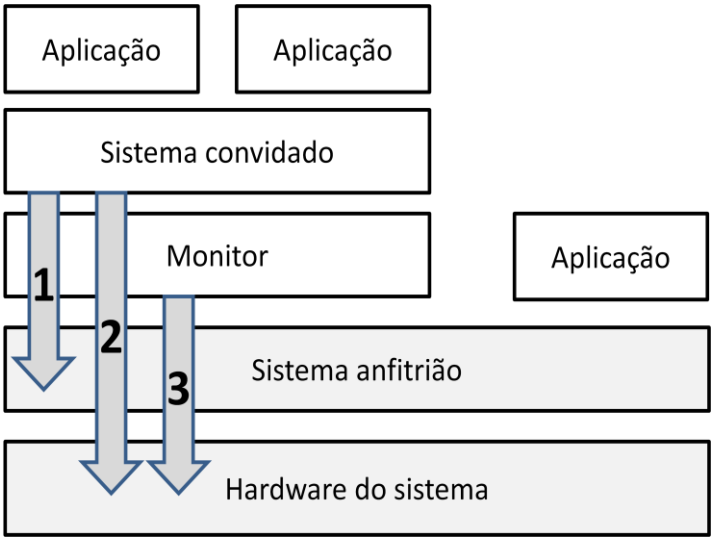


Figura 12 - Otimizações no modelo hospedado de VM de sistema (Laureano, 2006)

Onde:

(1) O sistema convidado acessa diretamente o sistema hospedeiro. Essa otimização é implementada no *hipervisor*, oferecendo partes da API do sistema hospedeiro ao sistema convidado, exemplo: VMWare pode utilizar o sistema de arquivos direto do hospedeiro, ao invés de virtualizar o sistema de arquivos.

(2) O Sistema convidado acessa diretamente o *hardware*. Essa otimização deve ser implementada no *hipervisor* e parcialmente pelo sistema hospedeiro, devido ao fato de poder usar algum *driver* específico, um exemplo é o acesso direto ao drive de CD implementado pelo VirtualPC e outros.

(3) O monitor acessa diretamente o *hardware*. Neste caso, um driver específico é instalado no sistema hospede para que o *hipervisor* possa ter acesso ao hardware.

2.8 Considerações finais

Este capítulo apresentou a evolução da virtualização nos sistemas computacionais. Mostrou alguns dos fatos principais que ajudaram a impulsionar o desenvolvimento desta tecnologia, como o fato da rápida absorção da arquitetura x86 nas empresas.

Através da massificação da computação nas empresas, foi necessário criar mecanismos que tirasse proveito do poder computacional que essas empresas possuíam subutilizadas. Assim a virtualização passou a ser uma ferramenta importante.

Com a evolução da virtualização, novos estudos foram desenvolvidos e novas técnicas de virtualização foram implementadas, como é o caso do conceito de máquina virtual, e suas propriedades, como foi apresentado ao longo deste capítulo.

No próximo capítulo serão discutidas de forma mais aprofundada as técnicas de virtualização.

Capítulo
3

Técnicas de Virtualização

Devido ao aumento dos computadores pessoais e o declínio dos computadores *mainframes*, a virtualização não foi o centro dos estudos durante a história da computação, pois não se via necessidade naquele momento de desenvolver a virtualização para computadores pessoais (Fisher-Ogden, 2007). Por causa disso e diferente dos *mainframes*, o x86 foi projetado não levando em consideração mecanismos que facilitassem a virtualização. Sendo assim, a arquitetura x86 não é dita totalmente apropriada para a virtualização clássica.

Entretanto, técnicas foram desenvolvidas para amenizar os problemas da arquitetura. Neste capítulo serão apresentados os desafios da arquitetura e algumas soluções para estes problemas.

3.1 Virtualização Clássica

Popek e Goldberg em 1974 definiram os requisitos para o termo virtualização clássica (Popek & Goldberg, 1974). Por esse padrão, uma peça de software pode ser considerada um VMM se ele atende os seguintes requisitos:

Execução equivalente. Programas sendo executados em um ambiente virtual devem se comportar identicamente como se estivesse sendo executada nativamente, com exceção em disponibilidade de recursos e *timing*.

Desempenho. Um subconjunto dominante de instruções deve ser executado nativamente na CPU.

Segurança. Um VMM precisa controlar completamente os recursos do sistema.

Uma técnica pioneira para virtualização foi o *trap-and-emulate* (tratado na seção 3.3). Enquanto esta técnica foi efetiva para prover um ambiente equivalente de execução, seu desempenho foi severamente atacado pois cada instrução poderia exigir dúzias de instruções para o VMM. Os requisitos de desempenho para o VMM não rejeita o *trap-and-emulate*, mas ao invés disso, limitam sua aplicação.

3.2 Desafios arquiteturais do x86

A arquitetura x86 suporta 4 níveis de privilégios, ou *rings*, com o *ring* 0 sendo o mais privilegiado e o *ring* 3 o com menos privilégios (Fisher-Ogden, 2007). Sistemas operacionais são executados no *ring* 0 pois geralmente necessitam ter acesso direto a memória e outros dispositivos, e precisa executar instruções privilegiadas, enquanto aplicações do usuário executam no *ring* 3, e *ring* 1 e 2 não são tipicamente usados. A figura 13, mostra esse conceito:

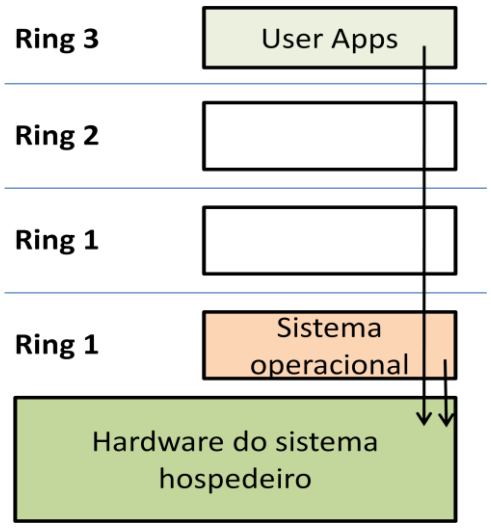


Figura 13 - Arquitetura x86 de privilégios (Citrix Inc., 2006)

Virtualizar a arquitetura x86 requer o posicionamento de uma camada de virtualização abaixo do sistema operacional (convidado), que se espera ser no *ring* mais privilegiado ou seja, o *ring 0*.

3.2.1 Ring Compression

Para prover isolamento entre as máquinas virtuais, o *hipervisor* (VMM) é executado no *ring* 0 e as máquinas virtuais no *ring* 1(modelo 0/1/3) ou *ring* 3 (modelo 0/3/3). Enquanto

o modelo 0/1/3 é mais simples, ele não pode ser usado quando executando no modo 64bit em uma CPU que suporta a extensão de 64bit na arquitetura x86.

Para proteger a VMM dos SO convidados, paginação ou limite de segmentação podem ser usado. Entretanto, limite de segmentação não é suportado no modo 64bit e paginação no x86 não distingue os *rings* 0,1 e 2. Isto resulta no *ring compression* onde um SO convidado precisa ser executado no *ring 3* desprotegido das aplicações dos usuários.

3.2.2 Ring Aliasing

Um problema relacionado é conhecido por *ring aliasing*, onde o verdadeiro nível de privilégio de um SO convidado é exposto, contrariando a crença do SO convidado de que está executando no *ring 0*. Por exemplo, executando a instrução PUSH no registrador CS, que inclui o nível de privilégio atual do sistema e em seguida examinando o resultado revelaria a discrepância do nível de privilégio do sistema.

3.2.3 Address Space Compression

Representa outra dificuldade para a virtualização na arquitetura x86. O VMM pode tanto executar em seu próprio espaço de endereçamento que pode ser custoso quando alternar entre os convidados e o VMM, ou pode executar em parte do espaço de endereço do convidado. Quando o VMM executa em seu próprio espaço de endereço, algum armazenamento no espaço de endereço convidado é ainda requisitado para estruturas de controle como a tabela de descritor de interrupção e a tabela de descritor global. Em ambos os casos, o VMM precisa proteger a porção de espaço de endereçamento que ele usa do convidado. Caso contrário, um convidado poderia descobrir que ele executa em um ambiente virtual ou comprometer o isolamento da máquina virtual lendo ou escrevendo nesses locais.

3.2.4 Instruções sensíveis não privilegiadas

Esta sendo como uma clara violação da virtualização clássica de Popek e Goldberg (Fisher-Ogden, 2007), a arquitetura x86 suporta instruções sensíveis que não são privilegiadas e desta maneira não causam um *trap* para a VMM realizar o tratamento adequado. Por exemplo, a instrução SMSW armazena o status da máquina em um registrador que pode ser lido por um sistema convidado (Robin & Irvine, 2000), expondo informações privilegiadas.

3.2.5 Falhas silenciosas de privilégio

Outro problema envolvendo estados privilegiados é que alguns acessos privilegiados, ao invés de causarem o *trap* para serem tratados pelo VMM, falham silenciosamente sem causar *faulting*. Isto viola o conceito que uma máquina virtual deve executar identicamente a uma execução nativa.

3.2.6 Interrupt Virtualization

Finalmente, *interrupt virtualization* pode ser um desafio para o as VM x86. O VMM quer gerenciar interrupções externas mascarando e desmascarando ele mesmo para manter o controle do sistema. Entretanto, alguns sistemas convidados freqüentemente mascaram e desmascaram interrupções, que resultaria em um desempenho pobre se uma troca para o VMM for requisitado em cada instrução mascarada.

3.3 Trap-and-emulate

A implementação clássica diz respeito as propriedades definidas por Popek e Goldberg, e é baseada no trap-and-emulate (Popek & Goldberg, 1974), cujo principais idéias são:

Desprivilegio. Numa arquitetura clássica virtualizável todas as instruções que lêem ou gravam um estado privilegiado e são executados num estado de desprivilegio geram um *trap*. Um *hipervisor* clássico executa um SO diretamente no computador, porém com um nível reduzido de privilégio. O *hipervisor* captura as *traps* geradas por instruções do SO que tenta acessar um estado privilegiado e emula estas instruções. Por exemplo, a execução de uma instrução desativando uma interrupção por sistema operacional convidado irá disparar um *trap*.

Estruturas sombreadas. Por definição, a VMM possui um estado de privilégio inferior ao da camada de hardware sob o qual executa. Para o sistema operacional convidado, esta diferença não deve existir e para atender as expectativas do sistema operacional, a VMM cria estruturas sombreadas derivadas das estruturas primárias do hardware. A CPU contém em seus registradores internos dados que estão dentro do estado de privilégio, por exemplo, o registrador para os ponteiros das tabelas de páginas ou o registrador de status do processador. Para criar uma estrutura sombreada, a VMM mantém uma imagem do registrador do sistema

convidado, e faz referência a essa imagem sempre que uma instrução for emulada após um *trap*. Entretanto, alguns dados fora da CPU e que são privilegiadas, como as tabelas de páginas, podem residir em memória. Neste caso, o acesso por parte do SO convidado através de uma estrutura sombreada pode não coincidir com as instruções de *trap* sobre a estrutura primária. Um exemplo são as entradas de uma tabela de paginação, que estão em memória e são de acesso privilegiado em função das permissões de acesso. Operações de I/O através de DMA podem tornar a estrutura sombreada do SO convidado inconsistente com a estrutura primária.

Rastreamento. Para manter coerência com as estrutura primárias, um mecanismo de "Rastreamento de Memória", ou *Memory Trace* é implementando em conjunto com as Estruturas de Sombra. As VMMs em geral utilizam mecanismos em hardware para proteção de paginação para realizar o *trap* de operações em estruturas de dados em memória. Por exemplo, as estruturas de sombra da PTE(*Page Table Entry*) do SO convidado derivadas das PTEs em memória são construídas com proteção de escrita, assim como dispositivos de E/S mapeados em memória também possuem suas páginas protegidas para leitura e escrita. Esta técnica de proteção de página é conhecida como *tracing* ou rastreamento. VMMs clássicas tratam uma exceção de *tracing* de forma similar a um *trap* de uma instrução privilegiada: decodifica a instrução do sistema operacional que gerou o *trap* e emula o seu efeito sobre a estrutura primária, propagando manualmente as alterações também na estrutura sombreada. A VMM usa o mecanismo de *trace* para se manter coerente com a PTE convidada. O resultado do uso deste mecanismo é uma grande fonte degradação de desempenho. Entretanto, deixar de aplicar esse oneroso mecanismo pode causar degradação ainda pior no desempenho, uma vez que diversas trocas de contexto seriam necessárias para validar as estruturas de sombreamento contra as estruturas primárias a cada acesso do SO convidado. Também, a ausência de qualquer controle seria uma fonte de "travamento por dependência de dados" por parte da MV, uma vez que ela teria que reconstruir a PTE a cada falha de acesso e só então prosseguir com a execução da MV.

3.4 Virtualização completa

Virtualização completa ou *full virtualization* pode virtualizar qualquer sistema operacional na arquitetura x86 usando uma combinação de tradução dinâmica e execução direta (Citrix Inc., 2006). Esta abordagem, mostrada na figura 14, traduz código no nível de

núcleo a fim de sobrescrever instruções não virtualizáveis com novas seqüências de instruções que tem o efeito desejado sobre o *hardware* virtual. Cada VMM provê para cada VM todos os serviços de um sistema físico, incluindo uma BIOS virtual, dispositivos virtuais e gerenciamento de memória virtual.

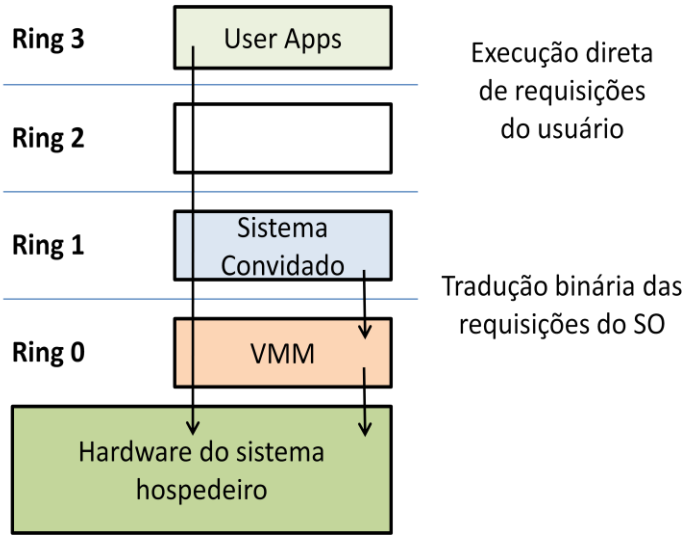


Figura 14 - Virtualização completa para a arquitetura x86 (Citrix Inc., 2006)

A figura 14 mostra esta combinação de tradução binária e execução direta provê virtualização completa, onde o SO convidado é completamente abstraído do hardware através da camada de virtualização. O SO convidado não compreende que está sendo virtualizado e não necessita de modificações. Virtualização completa é a única opção que não requer assistência via hardware ou assistência do sistema operacional para virtualizar instruções sensíveis e privilegiadas. O hipervisor (VMM) traduz todas as instruções do sistema operacional sucessivamente e armazena os resultados em um *cache* para uso futuro, enquanto instruções no nível de usuário executam inalterada com velocidade nativa.

Virtualização completa oferece o melhor isolamento e segurança para máquinas virtuais, e simplifica a migração e portabilidade, pois o mesmo SO convidado pode ser executado virtualizado ou de forma nativa. A virtualização usada pela VMWare® e Microsoft® Virtual Server® são exemplos de virtualização completa.

3.5 Paravirtualização

O prefixo “para-“ vem da origem grega que quer dizer “ao lado, junto”. Paravirtualização refere-se a comunicação entre o SO convidado e o *hipervisor* para melhorar desempenho e eficiência. Paravirtualização envolve modificação do núcleo do SO para substituir instruções não virtualizáveis com hiperchamadas que se comunicam diretamente com a camada de virtualização, o *hipervisor*, como mostra a figura 15.

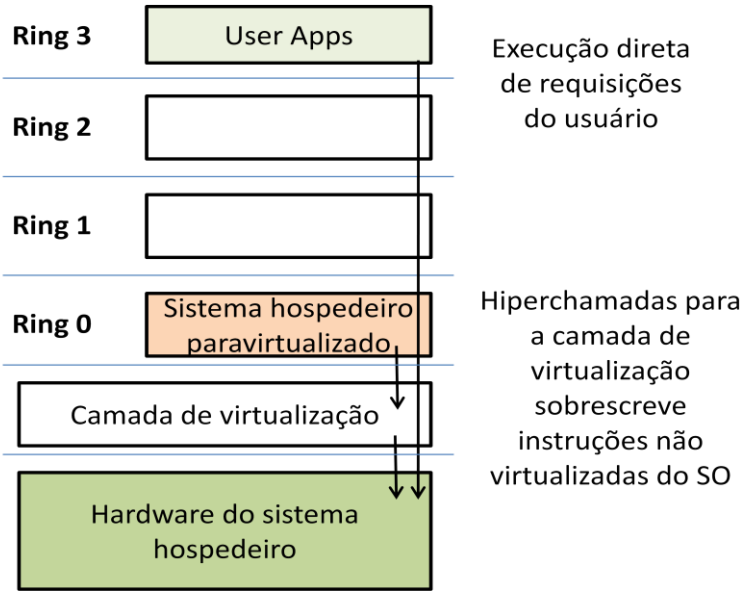


Figura 15 - Paravirtualização ou assistência via SO (Citrix Inc., 2006)

O *hipervisor* também provê interfaces para operações críticas como gerenciamento de memória, tratamento de interrupções e etc. Paravirtualização é diferente da virtualização completa, onde o SO não modificado não percebe que está sendo virtualizado e chamadas sensíveis do SO são tratadas usando tradução binária. A proposta da paravirtualização está no baixo *overhead*, mas o desempenho da paravirtualização sobre a virtualização completa pode variar grandemente de acordo com a carga de trabalho.

Como a paravirtualização precisa modificar o núcleo do SO, isso exclui sistemas operacionais não modificáveis (ex. Windows XP/Vista), sua compatibilidade e portabilidade são pobres. Paravirtualização pode também introduzir suporte significativo e manutenabilidade na produção de ambientes com requisitos de modificação do núcleo de SO. O projeto open source Xen é um exemplo de paravirtualização, que virtualiza o processador e memória usando um núcleo Linux modificado.

Enquanto é muito difícil construir uma tradução binária mais sofisticada necessária para a virtualização completa, modificar o SO convidado para habilitar paravirtualização é relativamente simples.

3.6 Tradução dinâmica

Uma técnica freqüentemente utilizada, inclusive utilizada na virtualização completa, é a tradução dinâmica (*dynamic translation*) ou recompilação dinâmica (*dynamic recompilation*) (Laureano, 2006). Nesta técnica o *hipervisor* analisa, reorganiza e traduz seqüências de instruções geradas pelo sistema convidado em novas seqüências de instruções.

A tradução binária dinâmica pode: (a) adaptar as instruções geradas pelo sistema convidado ao conjunto ISA do sistema real, caso não sejam idênticas; (b) detectar e tratar instruções sensíveis não-privilegiadas (que não geram interrupções ao serem invocadas pelo SO convidado); e (c) analisar, reorganizar e aperfeiçoar as seqüências de instruções de forma a melhorar o desempenho. Neste caso a utilização de *cache* é freqüente.

A tradução dinâmica poderá ser (Adams & Agesen, 2006):

Binária. A entrada é código binário x86, e não código fonte.

Dinâmico. A tradução acontece em tempo de execução, intercalando com a execução do código gerado.

On demand. Código é traduzido somente quando está para ser executado. Esta execução passo-a-passo elimina a necessidade de estruturas para armazenar código e dados.

Subsetting. A entrada do tradutor é completada com um conjunto de instruções x86, incluindo todas as instruções privilegiadas; a saída do tradutor é composta por um conjunto seguro de instruções, geralmente instruções no modo usuário.

Adaptativa. O código traduzido é ajustado em reposta a mudanças de comportamento do convidado para melhorar a eficiência de modo geral.

A tradução dinâmica é utilizada em vários tipos de *hipervisors*. Uma aplicação típica já comentada neste trabalho é a máquina virtual Java®, que recebe o nome de JIT – *Just in time Bytecode Compiler*. O uso corrente é a construção de *hipervisors* para plataformas sem

suporte adequado a virtualização, como os processadores Intel/AMD 32 Bits x86. Nestes as instruções sensíveis são substituídas por chamadas a rotinas apropriadas.

A recompilação dinâmica é composta dos seguintes passos (Ung & Cifuentes, 2006):

***Desmontagem* (*disassembling*)**: o fluxo de bytes do código convidado a executar é decomposto em blocos de instruções. Cada bloco é normalmente composto de uma seqüência de instruções de tamanho variável, terminando com uma instrução de controle de fluxo de execução;

Geração de código intermediário: cada bloco de instruções tem sua semântica descrita através de uma representação independente de máquina;

Otimização: a descrição em alto nível do bloco de instruções é analisada para aplicar eventuais otimizações; somente otimizações com baixo custo computacional são aplicáveis;

***Codificação*:** o bloco de instruções otimizado é traduzido para instruções da máquina física, que podem ser diferentes das instruções do código original;

Caching: blocos de instruções com execução muito freqüente têm sua tradução armazenada em *cache*;

***Execução*:** o bloco de instruções traduzido é finalmente executado nativamente pelo processador da máquina real.

O processo de tradução é simplificado quando as instruções de máquina do convidado são as mesmas da máquina real, o que torna desnecessário traduzir as instruções.

3.7 Virtualização assistida por hardware

Fornecedores de hardware estão rapidamente abraçando a virtualização e desenvolvendo novas características para simplificar as técnicas de virtualização. A primeira geração de melhorias inclui a Intel Virtualization Technology (VT-x) e AMD-V, em que ambos miram em instruções privilegiadas com novo modo de execução da CPU que permite que o *hipervisor* seja executado em um novo modo raiz abaixo no *ring* 0, como mostra a figura 16.

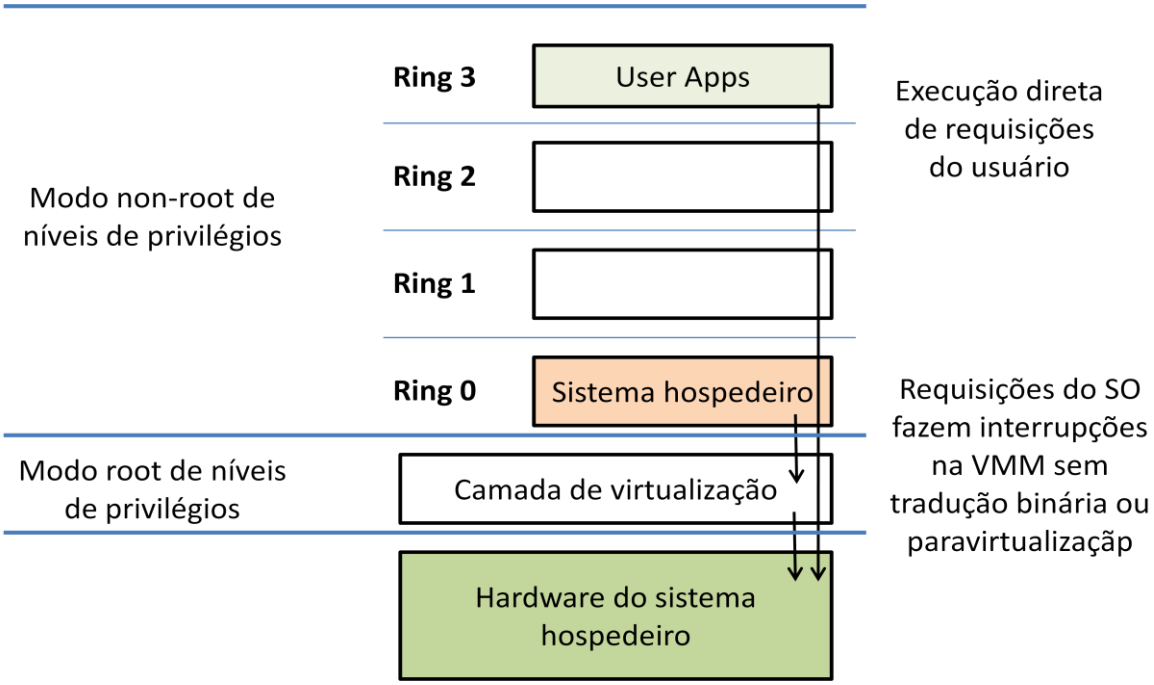


Figura 16 - Virtualização com assistência via hardware (Citrix Inc., 2006)

Chamadas privilegiadas e sensíveis são configuradas para automaticamente causarem um *trap* para o *hipervisor*, removendo a necessidade de tradução binária ou paravirtualização. O estado do convidado é armazenado na Virtual Machine Control Structures(VT-x) ou Virtual Machine Control Blocks (AMD-V). Processadores com Intel VT e AMD-V foram desenvolvidos a partir de 2006.

3.8 Virtualização de memória

Depois da virtualização do processador, o próximo componente crítico é a virtualização da memória. Isto envolve compartilhar a memória física do sistema e alocá-la dinamicamente para as máquinas virtuais. A virtualização da memória é muito similar a memória virtual providos pelos sistemas operacionais modernos. Aplicações enxergam um espaço de endereçamento contínuo que não necessariamente corresponde ao endereço físico. O Sistema operacional mantém um mapa das páginas virtuais para as páginas físicas. Todas as CPUs x86 modernas incluem uma unidade de gerenciamento de memória (MMU – Memory Management Unit) para aperfeiçoar o desempenho da memória virtual.

Para executar múltiplas máquinas virtuais em um mesmo sistema, outro nível de memória é exigido. Em outras palavras, deve-se virtualizar o MMU para dar suporte ao SO

convidado, o qual continua a controlar o mapeamento do endereço virtual para a memória convidada, mas ele não tem acesso direto à memória da máquina. O *hipervisor* é responsável por mapear a memória física da máquina.

O mapeamento para a memória virtual pode ser feita diretamente sobre a memória da máquina para evitar os dois níveis de tradução em todo o acesso. Quando o SO convidado atualiza sua tabela de paginas virtuais de memória, o hipervisor atualiza também suas referencias para a memória física. A virtualização do MMU cria alguns *overhead* para todas as abordagens feitas até o momento, mas esta é a área onde a segunda geração de virtualização com assistência via hardware irá oferecer maiores benefícios.

O mesmo processo ocorre para acesso a disco: enquanto num sistema com virtualização total o sistema convidado enxerga o disco como seu para uso exclusivo, sempre que o sistema realiza uma chamada de sistema para acesso a disco, o monitor de máquinas virtuais deve capturar essa chamada, interpretá-la e repassá-la para que o sistema anfitrião execute a operação. Na paravirtualização, o sistema convidado "sabe" que o disco é compartilhado e realiza diretamente o acesso ao disco (sem a interferência do monitor de máquinas virtuais), bem como aos demais dispositivos (mouse, teclado etc.); o monitor somente controla a ordem de acesso, no caso de múltiplos sistemas convidados em execução simultaneamente (Laureano, 2006).

3.9 Virtualização de dispositivo

O componente final exigido para a virtualização depois da CPU e memória são os dispositivos de I/O. Isto envolve gerenciamento de rotinas de requisições de I/O entre os dispositivos virtuais e os dispositivos físicos compartilhados.

Virtualização e gerenciamento de I/O baseado em software, em contraste com a passagem direta para o hardware, permitem um conjunto rico de características de gerenciamento. Com o dispositivo de rede, por exemplo, switches virtuais criam uma rede virtual entre a máquina virtual e o sistema hospedeiro sem haver consumo de banda na rede física.

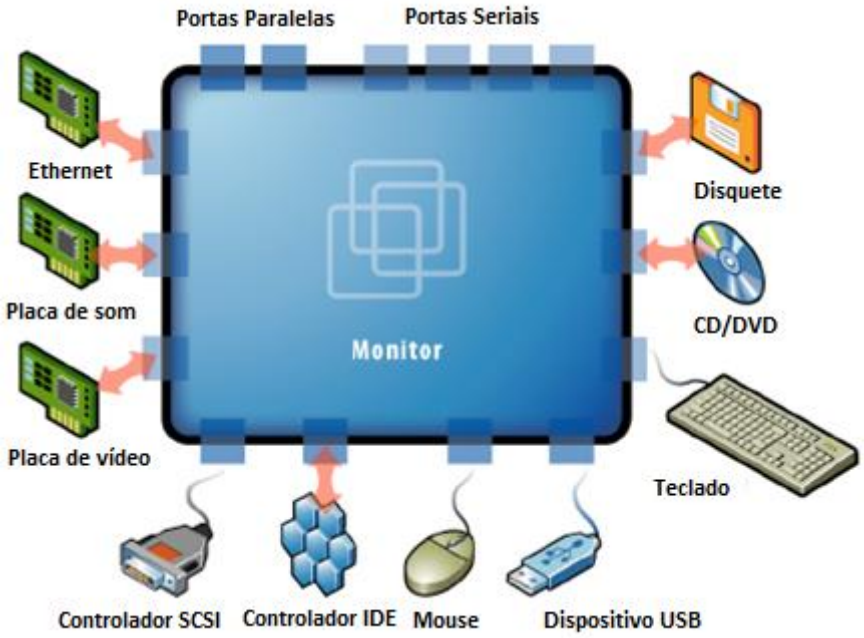


Figura 17 - Virtualização de I/O (Citrix Inc., 2006)

O *hipervisor* virtualiza o *hardware* físico e apresenta para cada máquina virtual um conjunto padronizado de dispositivos virtuais como visto na figura 17. Estes dispositivos virtuais efetivamente emulam o hardware e traduzem as requisições das máquinas virtuais para o sistema físico de hardware. Esta padronização através de consistentes *drivers* de dispositivos também ajuda na padronização da maquina virtual e na portabilidade através das plataformas, pois todas são configuradas para serem executadas sobre o mesmo hardware esquecendo-se do verdadeiro hardware físico do sistema.

3.10 VMWare

Atualmente, o *VMware* é a máquina virtual mas difundida para a plataforma x86, provendo uma implementação completa da interface x86 ao sistema convidado. Embora essa interface seja extremamente genérica para o sistema convidado, acaba conduzindo a um *hipervisor* mais complexo. Como podem existir vários sistemas operacionais em execução sobre mesmo hardware, o *hipervisor* tem que emular certas instruções para representar corretamente um processador virtual em cada máquina virtual, fazendo uso intensivo dos mecanismos de tradução dinâmica (VMWare, Inc, 2000).

Atualmente, a *VMWare* produz vários produtos com *hipervisors* nativos e convidados:

Hipervisor convidado:

VMware Workstation: primeira versão comercial da máquina virtual, lançada em 1999, para ambientes *desktop*;

VMware Fusion: versão experimental para o sistema operacional *Mac OS* com processadores Intel;

VMware Player: versão gratuita do *VMware Workstation*, com as mesmas funcionalidades mas limitado a executar máquinas virtuais criadas previamente com versões comerciais;

VMWare Server: conta com vários recursos do *VMware Workstation*, mas é voltado para pequenas e médias empresas;

Hipervisor nativo:

VMware ESX Server: para servidores de grande porte, possui um núcleo proprietário chamado *vmkernel* e Utiliza o *Red Hat Linux* para prover outros serviços, tais como a gerência de usuários.

O *VMware Workstation* utiliza as estratégias de virtualização total e tradução dinâmica. O *VMware ESX Server* implementa ainda a paravirtualização. Por razões de desempenho, o *hipervisor* do *VMware* utiliza uma abordagem híbrida para implementar a interface do hipervisor com as máquinas virtuais (Sugerman, Venkitachalam, & Lim, 2001). O controle de exceção e o gerenciamento de memória são realizados por acesso direto ao hardware, mas o controle de entrada/saída usa o sistema hospedeiro. Para garantir que não ocorra nenhuma colisão de memória entre o sistema convidado e o real, o *hipervisor VMware* aloca uma parte da memória para uso exclusivo de cada sistema convidado.

Para controlar o sistema convidado, o *VMware Workstation* intercepta todas as interrupções do sistema convidado. Sempre que uma exceção é causada no convidado, é examinada primeira pelo hipervisor. As interrupções de I/O são remetidas para o sistema hospedeiro, para que sejam processadas corretamente. As exceções geradas pelas aplicações no sistema convidado (como as chamadas de sistema, por exemplo) são remetidas para o sistema convidado.

3.11 FreeBSD Jails

O sistema operacional *FreeBSD* oferece um mecanismo de confinamento de processos denominado *Jails*, criado para aumentar a segurança de serviços de rede. Esse mecanismo consiste em criar domínios de execução distintos, denominados *jails* ou celas. Cada cela contém um subconjunto de processos e recursos (arquivos, conexões de rede) que pode ser gerenciado de forma autônoma, como se fosse um sistema separado (Kamp & Watson, 2000).

Cada domínio é criado a partir de um diretório previamente preparado no sistema de arquivos. Um processo que executa a chamada de sistema *jail* cria uma nova cela e é colocado dentro dela, de onde não pode mais sair, nem seus filhos. Além disso, os processos em um domínio não podem:

- Reconfigurar o núcleo (através da chamada sysctl, por exemplo);
- Carregar/retirar módulos do núcleo;
- Mudar configurações de rede (interfaces e rotas);
- Montar/desmontar sistemas de arquivos;
- Criar novos *devices*;
- Realizar modificações de configurações do núcleo em tempo de execução;
- Acessar recursos que não pertençam ao seu próprio domínio.

Essas restrições se aplicam mesmo a processos que estejam executando com privilégios de administrador (*root*). Pode-se considerar que o sistema *FreeBSD Jails* virtualiza somente partes do sistema hospedeiro, como a árvore de diretórios (cada domínio tem sua própria visão do sistema de arquivos), espaços de nomes (cada domínio mantém seus próprios identificadores de usuários, processos e recursos de IPC) e interfaces de rede (cada domínio tem sua interface virtual, com endereço IP próprio). Os demais recursos (como as instruções de máquina e chamadas de sistema) são preservados, ou melhor, podem ser usadas diretamente.

Essa virtualização parcial demanda um custo computacional muito baixo, mas exige que todos os sistemas convidados executem sobre o mesmo núcleo.

3.12 Xen

O ambiente *Xen* é um *hipervisor* nativo para a plataforma *x86* que implementa a paravirtualização. Ele permite executar sistemas operacionais como Linux especialmente modificado para executar sobre o *hipervisor* (Barham *et al*, 2003). Versões mais recentes do sistema *Xen* utilizam o suporte de virtualização disponível nos processadores atuais, o que torna possível a execução de sistemas operacionais convidados sem modificações (como o Windows, por exemplo), embora com um desempenho ligeiramente menor que no caso de sistemas paravirtualizados. Conforme seus criadores, o custo e impacto das alterações nos sistemas convidados são baixos e a diminuição do custo da virtualização compensa essas alterações (a degradação média de desempenho observada em sistemas virtualizados sobre a plataforma *Xen* não excede 5%).

As principais modificações impostas pelo ambiente *Xen* a um sistema operacional convidado são:

- O mecanismo de entrega de interrupções passa a usar um serviço de eventos oferecido pelo *hipervisor*; o núcleo convidado deve registrar um vetor de tratadores de exceções junto ao *hipervisor*;

- As operações de entrada/saída de dispositivos são feitas através de uma interface simplificada, independente de dispositivo, que usa *buffers* circulares de tipo produtor/consumidor;

- O núcleo convidado pode consultar diretamente as tabelas de segmentos e páginas da memória usada por ele e por suas aplicações, mas as modificações nas tabelas devem ser solicitadas ao *hipervisor*;

- O núcleo convidado deve executar em um nível de privilégio inferior ao do *hipervisor*;

- O núcleo convidado deve implementar uma função de tratamento das chamadas de sistema de suas aplicações, para evitar que elas tenham de passar pelo *hipervisor* antes de chegar ao núcleo convidado.

Como o *hipervisor* deve acessar os dispositivos de hardware, ele deve dispor dos *drivers* adequados. Já os núcleos convidados não precisam de *drivers* específicos, pois eles acessam dispositivos virtuais através de uma interface simplificada. Para evitar o desenvolvimento de *drivers* específicos para o hipervisor, o ambiente *Xen* usa uma abordagem alternativa: a primeira máquina virtual (chamada VM_0) pode acessar o hardware diretamente e provê os *drivers* necessários ao hipervisor. As demais máquinas virtuais (VM_i, $i > 0$) acessam o hardware virtual através do hipervisor, que usa os *drivers* da máquina VM_0 conforme necessário. Essa abordagem, apresentada na figura 18Figura 18 - Arquitetura do hipervisor Xen, simplifica muito a evolução do *hipervisor*, por permitir utilizar os *drivers* desenvolvidos para o sistema Linux.

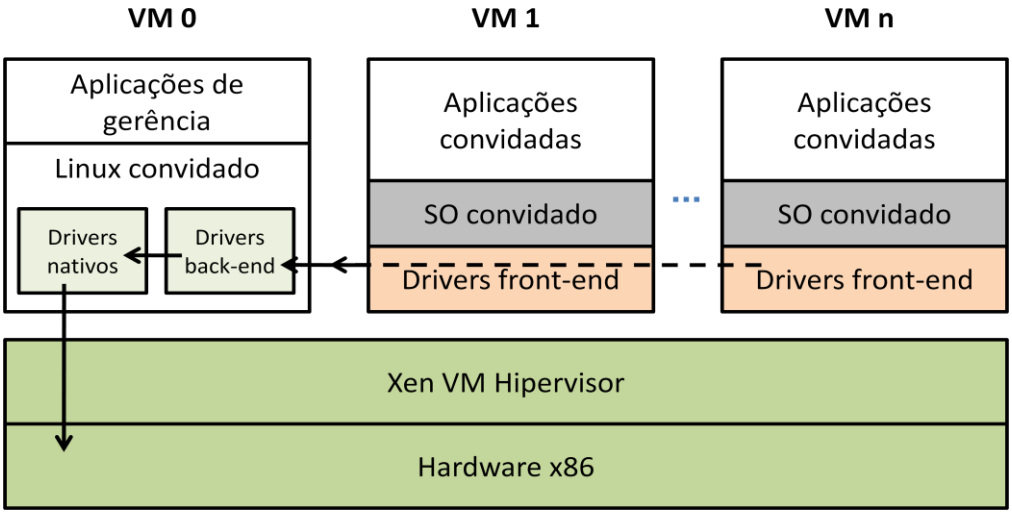


Figura 18 - Arquitetura do hipervisor Xen (Laureano, 2006)

O *hipervisor Xen* pode ser considerado uma tecnologia madura, sendo muito utilizado em sistemas de produção. O seu código-fonte está liberado sob a licença *GNU General Public Licence* (GPL). Atualmente, o ambiente *Xen* suporta os sistemas Windows, Linux e NetBSD. Várias distribuições Linux já possuem suporte nativo ao *Xen*.

3.13 User-Mode linux

O *User-Mode Linux* foi proposto por Jeff Dike em 2000, como uma alternativa de uso de máquinas virtuais no ambiente Linux (Dike, 2000). O núcleo do Linux foi portado de forma a poder executar sobre si mesmo, como um processo do próprio Linux. O resultado é um *user space* separado e isolado na forma de uma máquina virtual, que utiliza dispositivos de hardware virtualizados a partir dos serviços providos pelo sistema hospedeiro. Essa máquina virtual é capaz de executar todos os serviços e aplicações disponíveis para o sistema hospedeiro. Além disso, o custo de processamento e de memória das máquinas virtuais *User-Mode Linux* é geralmente menor que aquele imposto por outros *hipervisors* mais complexos.

O *User-Mode Linux* é hipervisor convidado, ou seja, executa na forma de um processo no sistema hospedeiro. Os processos em execução na máquina virtual não têm acesso direto aos recursos do sistema hospedeiro.

Como a máquina virtual é um processo no sistema hospedeiro, a troca de contexto entre duas instâncias de máquinas virtuais é rápida, assim como a troca entre dois processos do sistema hospedeiro. Entretanto, modificações no sistema convidado foram necessárias para a otimização da troca de contexto. A virtualização das chamadas de sistema é implementada pelo uso de uma *thread* de rastreamento que intercepta e redireciona todas as chamadas de sistema para o núcleo virtual. Este identifica a chamada de sistema e os seus argumentos, cancela a chamada e modifica estas informações no hospedeiro, onde o processo troca de contexto e executa a chamada na pilha do núcleo.

O *User-Mode Linux* está disponível na versão 2.6 do núcleo Linux, ou seja, ele foi assimilado à árvore oficial de desenvolvimento do núcleo, portanto melhorias na sua arquitetura deverão surgir no futuro, ampliando seu uso em diversos contextos de aplicação.

3.14 QEMU

O *QEMU* é um *hipervisor* com virtualização completa. Não requer alterações ou otimizações no sistema hospedeiro, pois utiliza intensivamente a tradução dinâmica como técnica para prover a virtualização. É um dos poucos *hipervisors* recursivos, ou seja, é possível chamar o *QEMU* a partir do próprio *QEMU*.

O hipervisor *QEMU* oferece dois modos de operação:

Emulação total do sistema: emula um sistema completo, incluindo processador e vários periféricos. Neste modo o emulador pode ser utilizado para executar diferentes sistemas operacionais;

Emulação no modo de usuário: disponível apenas para o sistema Linux. Neste modo o emulador pode executar processos Linux compilados em diferentes plataformas (por exemplo, um programa compilado para um processador *x86* pode ser executado em um processador *PowerPC* e vice-versa).

Por meio de um módulo instalado no núcleo do sistema hospedeiro, denominado *KQEMU* ou *QEMU Accelerator*, o *hipervisor QEMU* consegue obter um desempenho similar ao de outras máquinas virtuais como *VMWare* e *User-Mode Linux*. Com este módulo, o *QEMU* passa a executar as chamadas de sistema emitidas pelos processos convidados diretamente sobre o sistema hospedeiro, ao invés de interpretar cada uma. O *KQEMU* permite associar os dispositivos de I/O e o endereçamento de memória do sistema convidado aos do sistema hospedeiro. Processos em execução sobre o núcleo convidado passam a executar diretamente no modo usuário do sistema hospedeiro. O modo núcleo do sistema convidado é utilizado apenas para virtualizar o processador e os periféricos.

O *VirtualBox* (Sun Inc., 2008) é um ambiente de máquinas virtuais construído sobre o hipervisor *QEMU*. Ele é similar ao *VMware Workstation* em muitos aspectos. Atualmente, pode tirar proveito do suporte à virtualização disponível nos processadores Intel e AMD. Originalmente desenvolvido pela empresa *Innotek*, o *VirtualBox* foi adquirido pela *Sun Microsystems* e liberado para uso público sob a licença *GPLv2*.

3.15 JVM – Java Virtual Machine

É comum a implementação do suporte de execução de uma linguagem de programação usando uma máquina virtual. Um exemplo atual dessa abordagem ocorre na linguagem Java. Tendo sido originalmente concebida para o desenvolvimento de pequenos aplicativos e programas de controle de aparelhos eletroeletrônicos, a linguagem Java mostrou-se ideal para ser usada na Internet. O que o torna tão atraente é o fato de programas escritos em Java poderem ser executados em praticamente qualquer plataforma.

A virtualização é o fator responsável pela independência dos programas Java do hardware e dos sistemas operacionais: um programa escrito em Java, ao ser compilado, gera um código binário específico para uma máquina abstrata denominada *máquina virtual Java* (*JVM - Java Virtual Machine*). A linguagem de máquina executada pela máquina virtual Java é denominada *bytecode* Java, e não corresponde a instruções de nenhum processador real. A máquina virtual deve então interpretar todas as operações do *bytecode*, utilizando as instruções da máquina real subjacente para executá-las. A vantagem mais significativa da abordagem adotada por Java é a *portabilidade* do código executável: para que uma aplicação Java possa executar sobre uma determinada plataforma, basta que a máquina virtual Java esteja disponível ali (na forma de um suporte de execução denominado JRE - *Java Runtime Environment*). Assim, a portabilidade dos programas Java depende unicamente da portabilidade da própria máquina virtual Java. O suporte de execução Java pode estar associado a um navegador Web, o que permite que código Java seja associado à páginas Web, na forma de pequenas aplicações denominadas *applets*, que são trazidas junto com os demais componentes de página Web e executam localmente no navegador. A figura 19 mostra os principais componentes da plataforma Java.

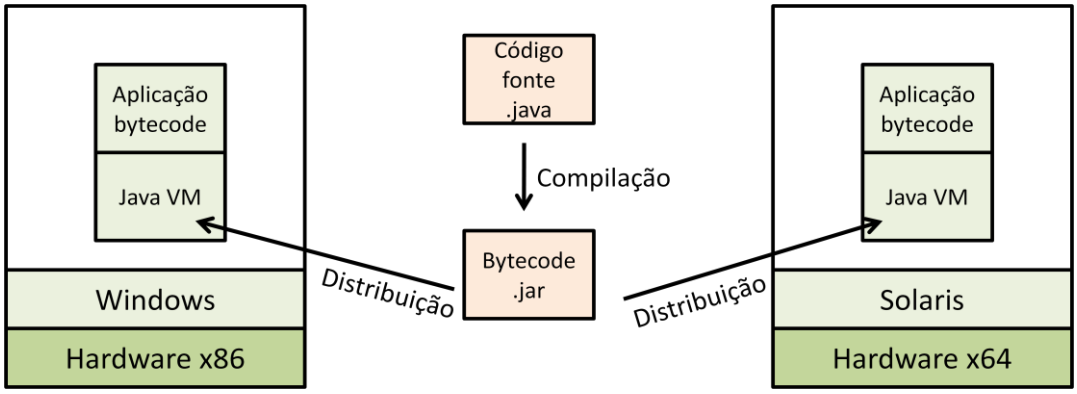


Figura 19 - Maquina virtual Java (Citrix Systems, Inc., 2006)

É importante ressaltar que a adoção de uma máquina virtual como suporte de execução não é exclusividade do Java, nem foi inventada por seus criadores. As primeiras experiências de execução de aplicações sobre máquinas abstratas remontam a década de 1970, com a linguagem *UCSD Pascal*. Hoje, muitas linguagens adotam estratégias similares, como Java, C#, Python, Perl, Lua e Ruby. Em C#, o código-fonte é compilado em um formato intermediário denominado CIL (*Common Intermediate Language*), que executa sobre uma

máquina virtual CLR (*Common Language Runtime*). CIL e CLR fazem parte da infra-estrutura .NET da Microsoft.

Em termos de desempenho, um programa compilado para uma máquina virtual executa mais lentamente que seu equivalente compilado sobre uma máquina real, devido ao custo de interpretação do *bytecode*. Todavia, essa abordagem oferece melhor desempenho que linguagens puramente interpretadas. Além disso, técnicas de otimização como a tradução dinâmica (ou compilação *Just-in-Time*), na qual blocos de instruções repetidos freqüentemente são compilados pelo monitor e armazenados em *cache*, permitem obter ganhos significativos de desempenho.

3.16 Considerações finais

Foram mostradas diversas técnicas de virtualização utilizadas pelas soluções e trabalhos atuais como mostrado nas ultimas seções do capítulo.

Podemos destacar os desafios que a arquitetura x86 apresenta para suportar a virtualização, e as soluções empregadas para atingir tal objetivo. Dentre as técnicas mais utilizadas, podemos destacar a chamada *trap-and-emulate* onde as instruções que precisam ser executadas de em modo de privilégio conseguem ser capturadas e tratadas.

Além disso foi apresentada a *virtualização completa* e as suas diferenças com relação a *paravirtualização,* que tem como principal característica a necessidade de modificação do SO convidado.

Os conceitos e mecanismos das técnicas apresentadas neste capítulo poderão ser aplicadas, com as devidas adaptações, na virtualização de aplicação que será discutida no próximo capítulo.

Capítulo
4

Virtualização de Aplicação

Visto o potencial deste seguimento da virtualização, algumas empresas deram início as primeiras soluções nesta área, e de acordo com estas soluções podemos chegar à algumas definições e a classificação da virtualização de aplicação. Nesta seção será apresentada a idéia que envolve a virtualização de aplicação com ou não streaming de software, apresentando exemplos reais de sistemas de virtualização de aplicação, bem como serão apresentados suas principais características que as diferenciam umas das outras.

4.1 Conceito de virtualização de aplicação

A idéia por trás da virtualização de aplicação pode ser entendida por alguns como relativamente simples, que é a execução de uma aplicação em um computador sem ter havido nenhum tipo de instalação nesta máquina. Entretanto essa visão simplista pode se misturar a conceitos existentes não diferenciando uma arquitetura de virtualização completa de aplicação. Diferente do Terminal Service por exemplo, que é um ambiente de desktop remoto que está sendo executado em um servidor especializado, a virtualização de aplicação consome os recursos locais da máquina em que está sendo utilizado. Estes recursos incluem processamento local, memória, espaço em disco, e dispositivo de rede, como mostra a figura 20.

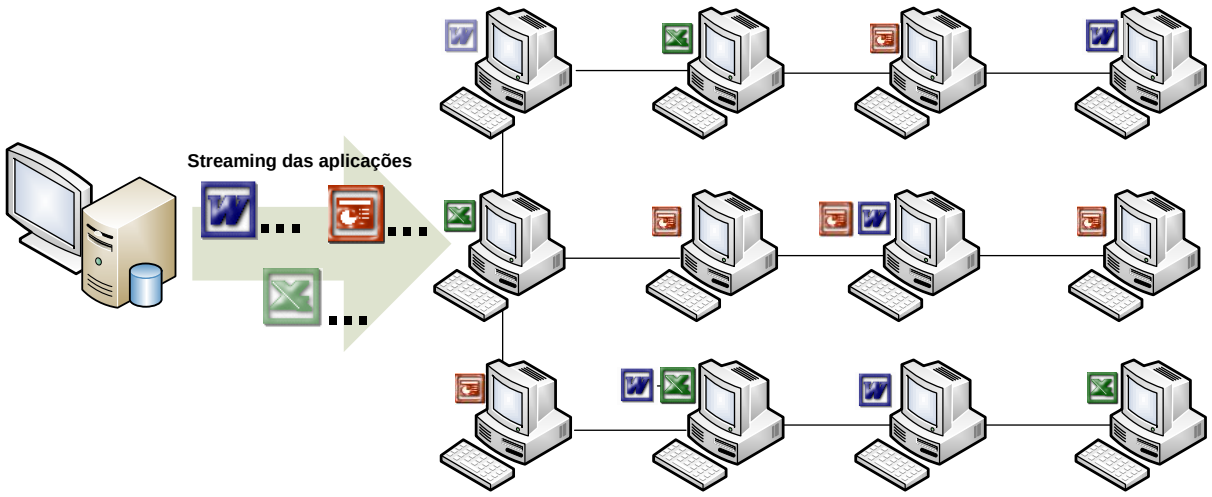


Figura 20 - Streaming de software. As aplicações podem ser executadas diretamente de um servidor de streaming, sem a necessidade de instalação

Em uma definição mais formal de virtualização de aplicação segundo uma empresa desta área (VMWare, 2010), a virtualização de aplicação permite a distribuição de software sem modificar o sistema operacional local ou o sistema de arquivos da máquina. Permite que software seja entregue em um ambiente isolado assegurando a integridade do sistema operacional hospedeiro e todas as outras aplicações contidas neste sistema. Problemas de conflitos, reconfigurações e desinstalação das aplicações são drasticamente reduzidos, trazendo a liberdade de testes sem comprometer o computador utilizado, porque o software na máquina é resumido a apenas um arquivo de imagem ou uma pasta única contendo tudo que a aplicação necessita para ser executada de forma virtualizada.

O conceito básico da virtualização clássica também se aplica na virtualização de aplicação. A aplicação virtualizada executa sobre a máquina local usando os seus recursos, mas sua execução não irá modificar o sistema hospedeiro. Ao invés disso, ele é executado em um pequeno ambiente virtual que contém arquivos necessários, entradas de registros, objetos COM, e outros recursos que ele necessita para ser executado. Este ambiente virtual atua como uma camada entre a aplicação e o sistema operacional. A camada virtual deve ser leve e geralmente ocupar pouco espaço em memória e ser carregado apenas no momento exato em que a aplicação a ser virtualizada será executada.

Diferente de uma máquina virtual onde um sistema operacional inteiro é instalado (junto com suas aplicações), a camada virtual ou middleware de virtualização reflete exatamente as características do sistema operacional, filtrando as ações que aplicação toma através das API que o sistema operacional oferece. Em outras palavras, uma máquina virtual provê uma abstração entre o hardware e o sistema operacional enquanto que um middleware de virtualização de aplicação provê uma abstração entre o sistema operacional e a aplicação em execução. A figura 21 mostra a diferença entre as máquinas virtuais (virtual machines) e a virtualização de aplicação e ainda, a diferença na utilização de terminais de serviço (terminal services).

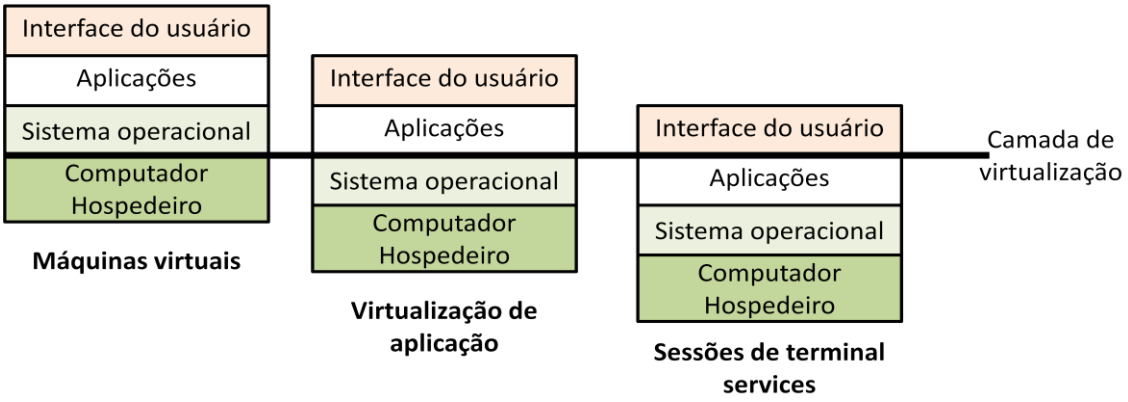


Figura 21 - Diferença entre máquinas virtuais, aplicações virtualizadas e interfaces virtuais (terminais de serviço)

4.2 Classificações da virtualização de aplicação

Existem dois tipos de virtualização de aplicação. Estes podem ser:

- ***Standalone*** onde a execução ocorre de forma independente e autônoma. Geralmente a aplicação é encapsulada em um único arquivo executável. Este executável pode ser executado instantaneamente de um dispositivo de memória, ou seja, um pendrive, CDROM ou disco de armazenamento local, ou ainda, podem ser distribuídos com download via rede, porém precisa que todo o pacote seja copiado para iniciar a execução da aplicação.

- ***Centralizado em servidor*** neste caso as aplicações são distribuídas através de um ambiente ou ferramenta centralizada em servidor. A aplicação não é empacotada em um único executável. Neste caso um middleware de virtualização é responsável

pelo gerenciamento de download do servidor dos recursos necessários para a execução da aplicação desejada no momento exato em que estes recursos são necessários.

É importante ressaltar que ambos os tipos apresentados acima podem oferecer o serviço de streaming, ou seja, podem ser distribuídos via uma rede comum de computadores, mas somente o modelo centralizado pode tornar eficiente o modo com que estas aplicações são carregadas. No modelo centralizado onde a execução é assistida pelo middleware de virtualização, os recursos são carregados *on-demand* tornando a distribuição e conseqüentemente a execução da aplicação mais rápida evitando gargalos de rede. Segundo um estudo feito pela própria Microsoft (Microsoft Softgrid, 2007) o carregamento de uma aplicação utiliza em média de 20% a 40% do total instalado no computador, sendo que o restante pode ser exigido em outros momentos durante a execução do software.

Segundo os tipos atuais das soluções existentes no mercado, podemos dividir a virtualização de aplicação nos seguintes tipos:

- ***Virtualização de Aplicação Server-based***: Executa a interface da aplicação no cliente e todo o processamento lógico fica no servidor. Um exemplo desta categoria são as aplicações sendo executadas em modo de terminal de serviços ou terminal services.

- ***Virtualização de aplicação Server-side*** : a aplicação continua sendo executada no servidor mas somente ela é encapsulada ou uma solução própria que permite resolver conflitos quando a aplicação é executada mais de uma vez ao mesmo tempo. Este ambiente virtual é capaz de isolar o processamento das aplicações protegendo umas das outras, sem precisar levantar outra seção de usuário novamente como é o caso do terminal service. Um exemplo desta categoria é o Citrix XenApp no modo hospedado.

- ***Virtualização de Aplicação Client-side:*** Utiliza a execução e o isolamento inteiramente no cliente. É o caso do ThinApp.

Embora o App-V utilize o processamento no lado cliente como veremos a seguir, ele utiliza parte do processamento do servidor que é responsável por seqüenciar sob demanda o

software sendo utilizado, mesmo que menor que o XenApp, o servidor possui uma porção de processamento a cada cliente adicionado. A figura 22 mostra os níveis de isolamento da virtualização de aplicação.

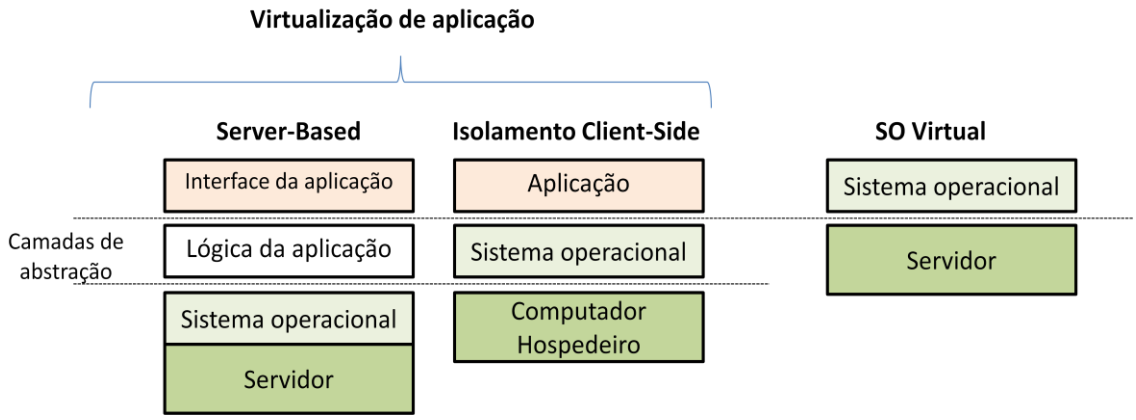


Figura 22 - Diferença entre o modelo baseado no servidor e o modelo de isolação no lado cliente

Na próxima seção serão abordados os exemplos citados. Podemos ainda perceber que o modelo ideal talvez misture um pouco das categorias apresentadas.

4.3 Microsoft App-V

Microsoft Application Virtualization (ou como era conhecido Microsoft Softgrid) é uma solução de virtualização de aplicação. Foi adquirida da empresa Softricity em Julho de 2006. (Microsoft-b, 2007).

O App-V da Microsoft (Microsoft Softgrid, 2007) é capaz de empacotar aplicações para a utilização sob demanda. Uma estação (computador) é separada para o módulo que irá fazer uma espécie de reconhecimento dos principais recursos utilizados pela aplicação. O administrador do App-V instala a aplicação desejada nesta estação, em seguida é monitorado todos os passos da instalação, incluindo alterações em registros e sistemas de arquivos. Após a coleta desses dados, quatro arquivos são disponibilizados para o servidor de streaming.

O administrador precisa garantir acesso de todos os usuários no Active Directory (AD) do domínio em que o servidor e os clientes estão inseridos para terem permissão de acessar as aplicações preparadas. Somente os usuários membros do grupo em questão poderão acessar as aplicações.

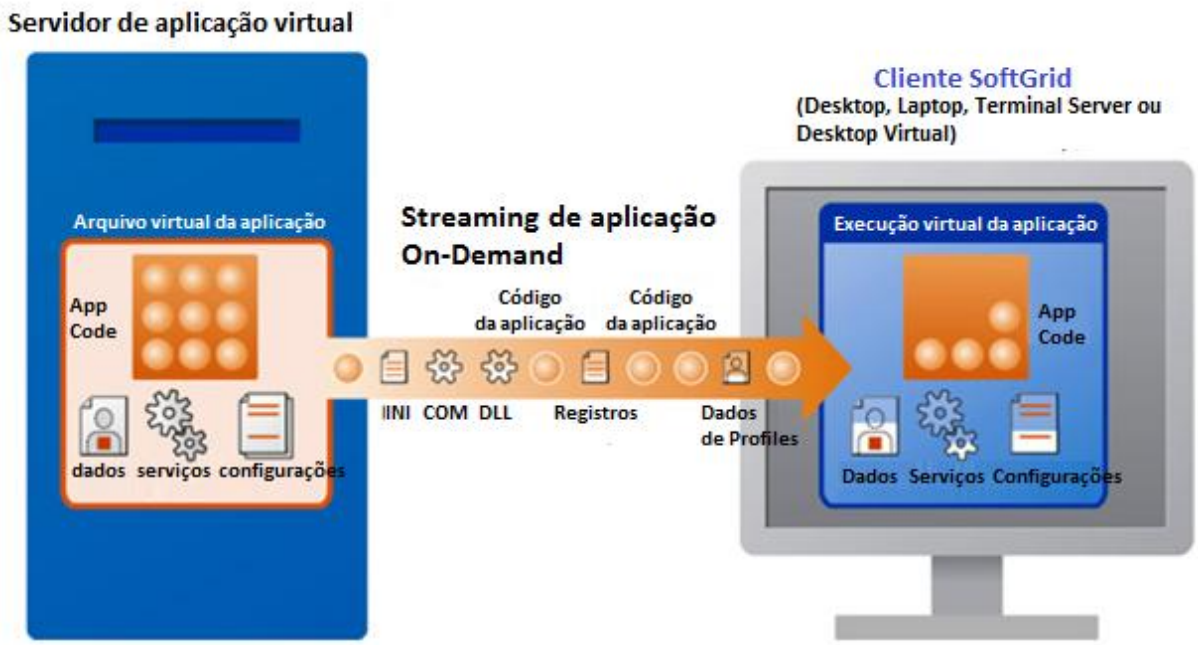


Figura 23 - Modelo Microsoft App-V de streaming de aplicação (Microsoft Softgrid, 2007)

Os quatro arquivos gerados pela instalação da aplicação no inicio do processo não são totalmente descarregados nas estações clientes. Ao longo da utilização do software o agente instalado no computador cliente requisita ao servidor os "trechos" necessários destes arquivos para continuar a execução, como mostrado na figura 23. Estes arquivos compõem registros, arquivos, DLL, objetos COM, arquivos INI, outros processos, fonts e etc.

Um mecanismo interessante aproveitado com melhorias pelo modelo proposto neste trabalho é o mecanismo de seqüenciamento do software, também chamado de ***App-V Sequencer***. Durante a instalação da aplicação, após todas as modificações no sistema operacional e no sistema de arquivos serem capturadas, o software alvo é lançado automaticamente. Neste momento o seqüenciador monitora os principais recursos utilizados que deverão ser enviados primeiro a estação cliente.

Qualquer atualização no sistema forçará que todos os pacotes entregues sejam novamente descarregados do servidor. Os pacotes enviados aos clientes ficam armazenados em um driver que é montado pelo agente do App-V instalado nas estações cliente. Isto requer privilégios especiais no cliente, ou seja, a aplicação precisa ser executada com direitos administrativos.

A arquitetura completa da solução App-V consiste nos seguintes componentes:

- **Microsoft Systems Center Virtual Application Server**, também chamado de **App-V Application Server,** o qual hospeda os pacotes das aplicações virtualizadas e realizam o streaming para os computadores clientes. Ele também pode requisitar que os clientes enviem logs do uso das aplicações. Como discutido as aplicações são convertidas em pacotes menores através do ***App-V Sequencer***.

- **Microsoft App-V Application Virtualization for Desktops,** também chamado de **App-V client**, é o agente que fica instalado na estação cliente do usuário. Ele é responsável pelas requisições ao servidor de streaming durante a execução das aplicações. Ao receber os pacotes principais, lança a o ambiente virtual ao qual a aplicação será executada.

- **App-V managemente console,** é a ferramenta de gerenciamento e administração que gerencia todos os servidores envolvidos na infra-estrutura do App-V. Nele são definidas as políticas de uso para o acesso as aplicações.

Para aumentar a eficiência da distribuição dos pacotes para os clientes, o App-V permite que a solução seja instalada em servidores diferentes ligados em forma de cluster com balanceamento de carga, sendo executado sobre um sistema de arquivos distribuído. Isto vem a dificultar a utilização desta solução por pequenas empresas ou segmentos, pois ao aumentar a quantidade de usuários, a infra-estrutura precisa aumentar proporcionalmente.

4.4 VMWare ThinApp

VMWare ThinApp (VMWare ThinApp, 2010), é um solução que possibilita o empacotamento de uma aplicação e todos os seus componentes em um único arquivo tornando as aplicações portáteis.

ThinApp é uma solução de virtualização de aplicação pois possibilita executar aplicações sem que elas sejam instaladas da maneira tradicional, inclusive permite que a mesma aplicação seja executada várias vezes de forma virtualizada, cada uma em seu ambiente virtual.

Ao empacotar a aplicação, a solução inclui os recursos necessários para sua execução como variáveis de ambiente, arquivos e chaves de registros além da camada virtual necessária para que a aplicação seja executada no computador cliente. Quando o arquivo final (que contém a aplicação e todos os recursos) é executado no computador cliente, primeiramente é lançado a camada de virtualização e em seguida a aplicação é executada neste ambiente virtual, assim todas as requisições que a aplicação realizar serão direcionadas aos recursos corretos.

Uma vantagem do ThinApp sobre o App-V é que ele não precisa que nenhum componente ou agente seja instalado previamente na estação cliente e também não precisa de privilégios administrativos para ser executado, mas por outro lado, não possui um mecanismo inteligente de seqüenciamento ou streaming do software virtualizado. Para que a execução via ThinApp aconteça, o pacote final completo precisa ser copiado para o computador cliente. Levando em consideração um software comercial de tamanho elevado com o Microsoft Office (que possui em média 500 Mbytes de instalação) o modelo do ThinApp não oferece grande vantagem, pois o pacote todo precisaria ser copiado antes da execução.

Muitas aplicações Windows podem ser empacotadas no modelo ThinApp, com a exceção de software que utilizem controle de cópia baseada em hardware. ThinApp não permite que alterações realizadas durante a execução sejam armazenadas de forma persistente. Na sua última versão, o ThinApp permite que o pacote seja quebrado em pacotes menores para ser enviados ao cliente, porém não mesmo assim todos eles precisarão ser sincronizados para que a aplicação possa ser executada.

4.5 Citrix XenApp

Citrix XenApp (Citrix XenApp, 2010)**,** formalmente conhecido como ***Citrix MetaFrame Server*** e ***Citrix Presentation Server***, permite que os usuários se conectem as suas aplicações corporativas. XenApp pode tanto hospedar as aplicações em servidores centrais e permitir que usuários as acessem como terminais de serviço remoto (terminal services) quanto pode entregar as aplicações aos usuários para execução local.

O núcleo e o ponto forte do XenApp é o uso de um protocolo proprietário na camada de apresentação (ou thin client protocol) chamado Arquitetura de Computação Independente (***ICA – Independent Computing Architecture***). Diferente de protocolos de *framebuffers*

como o VNC (Virtual networtk Computing), o ICA transmite informações de alto-nível das janelas da aplicação diferente dos demais protocolos onde a informação basicamente é gráfica. Um driver cliente é instalado nas máquinas para que capture os comandos do usuário sobre o display da aplicação transmitida para que possam ser reproduzidas no espaço virtualizado do servidor, interagindo dessa forma no sistema.

Já no modo local, o XenApp basicamente se comporta da mesma maneira do ThinApp. Ele disponibiliza um pacote único com todos os recursos da aplicação mais o middleware responsável por criar o ambiente virtual durante a execução *standalone*.

Os componentes do XenApp, incluindo as aplicações servidoras, podem ser hospedados em servidores com Microsoft Windows, que podem fazer parte de um conjunto de servidores (cluster) para o balanceamento de carga (**NLB – Network Load Balance**) já que no modo mais utilizado (o modo hospedado) o processamento é todo concentrado no servidor, e a cada entrada de um novo usuário, a carga dos servidores aumenta.

Como no caso do ThinApp, o XenApp não possui um sistema aprimorado de predição dos recursos iniciais a serem utilizados pela aplicação. Tal como o ThinApp e ao contrário do App-V, possui suporte para execução a partir de mídias diversas como *pendrives*, CDROM e disco local. Possui suporte off-line deste que o pacote único do software tenha sido copiado por completo na estação cliente.

4.6 Considerações finais

Mesmo sendo um conceito relativamente recente, comparando com a concepção do conceito de virtualização, a virtualização de aplicação tem cada vez mais entrado de fato na organização da infra-estrutura das empresas e parque de máquinas.

Este capítulo discutiu os conceitos envolvidos na virtualização de aplicação, mostrando os diferentes tipos e classificações das soluções empregadas para tal. A seguir mostrou as diferentes arquiteturas empregadas para virtualizar aplicações, discutindo o funcionamento de soluções reais. Como discutido ao longo do capítulo, os conceitos básicos da virtualização clássica também podem ser adaptados para o contexto das aplicações.

Capítulo
5

Redes peer-to-peer

O objetivo deste capítulo é apresentar conceitos básicos das redes peer-to-peer, bem como apresentar a classificação que estas redes recebem de acordo com sua arquitetura. Tais conceitos são importantes para o modelo ao qual este trabalho se propõe, pois utiliza, mesmo sendo o modelo básico, uma rede peer-to-peer para distribuir o software entre os participantes da rede.

5.1 Introdução as redes peer-to-peer

Uma rede peer-to-peer, comumente chamada de P2P, pode ser definida de várias maneiras, mas é, em sua essência, qualquer rede com arquitetura distribuída composta de participantes que oferecem uma parte de seus recursos que podem ser processamento, energia, armazenamento, banda de rede, etc., diretamente para outros participantes da rede, sem a necessidade de um coordenador central atuar junto a cópia do recurso, como um servidor, por exemplo (Schollmeier, 2002). Os participantes são, ao mesmo tempo, fornecedores e consumidores de recursos uns dos outros. Isto contrasta diretamente com o modelo cliente-servidor tradicional, onde somente os servidores disponibilizam seus recursos e os clientes somente os consomem.

Apesar de encontrarmos na literatura outras definições baseadas em modelos de interconexão, percebe-se que muitos modelos são rotulados como P2P não devido a sua arquitetura ou o modo que operam, mas pela impressão que ele passa por estar conectando computadores diretamente. Desta forma, diversas definições de P2P são aplicadas para acomodar os diferentes sistemas e aplicações P2P existentes. Em (Androutsellis-Theotokis & Spinellis, 2004) são apresentadas duas características que definem os sistemas P2P:

- Compartilhamento de recursos computacionais de forma direta entre as partes envolvidas, sem intermédio de servidores centrais, no entanto, se aceita a existência de servidores centrais para a execução de tarefas específicas, como a localização dos parceiros e autenticação de usuários.

- Habilidade para tratar instabilidade e variação de conectividade, adaptando-se de forma automática a falhas nas conexões de rede e computadores, bem como comportamento transitório dos parceiros.

De acordo com estas características, chegamos a definição que sistemas P2P são sistemas distribuídos, constituídos por parceiros interconectados, habilitados para a auto-organização em uma topologia de rede, com o propósito de compartilhar recursos como conteúdo, ciclos de CPU, armazenamento e largura de banda, com capacidade de se adaptar a falhas e acomodar-se à variação no número de participantes, mantendo aceitável conectividade e desempenho, sem a necessidade de intermediação ou suporte de um servidor global centralizado.

Como o intuito deste capítulo é apresentar de forma objetiva as redes peer-to-peer, suas características e sua classificação, não nos deteremos a discutir modelagem de algoritmos ou conceitos aprofundados sobre este tema, pois o modelo proposto neste trabalho utiliza sua forma mais simples dentro do conceito mais básico de rede peer-to-peer visto anteriormente, que são computadores servindo e consumindo recursos uns dos outros, de forma a não sobrecarregar o servidor e participantes em específicos da rede.

5.2 Características das redes peer-to-peer

Como mostrado em (Hu, 2005), as principais características das redes P2P, são:

A) Descentralização. O modelo de redes P2P vem questionar a centralização e o acesso a recursos em servidores através de protocolos de requisição/resposta. Apesar de este modelo centralizado reduzir o custo de hardware, tal modelo traz desvantagens como surgimento de gargalos no sistema. A idéia da descentralização está diretamente relacionada a posse de recursos por parte dos usuários. Em sistemas totalmente descentralizados, cada nó é um participante igual aos outros. Porém, devido ao aumento na complexidade de implementação e

gerenciamento dos sistemas totalmente descentralizados, alguns sistemas P2P acabam utilizando uma solução híbrida, em que existe um servidor com informações sobre o caminho para a troca de informações que é realizada diretamente entre os nós participantes

B) Escalabilidade. Na arquitetura cliente/servidor, todos os acessos ficam sobre os servidores que são os únicos responsáveis por fornecer os recursos, isto pode fazer com que hajam gargalos quando vários clientes fazem o aceso ao recurso ao mesmo tempo. No modelo P2P, o aumento do número de clientes é diretamente proporcional ao número de servidores, porque sendo um cliente também um fornecedor de recursos, na medida em que novos clientes entram na rede, ao mesmo tempo se tornam os novos "servidores", uma vez que todos os participantes atuam como cliente e servidores ao mesmo tempo. Desta forma, a disponibilidade dos recursos aumenta na mesma proporção do aumento da demanda, isto faz com que o sistema cresça sem comprometer sua capacidade de atender a qualidade de serviço oferecida e sem gerar gargalos.

C) Auto-organização. A habilidade de um sistema se manter organizado espontaneamente sem a necessidade de um sistema externo para é chamado de auto-organização. Em conjunto com a escalabilidade, a auto-organização se torna necessário para recuperação de falhas e conexões intermitentes. Em um sistema P2P por definição não se pode prever o número final de participantes, assim, um sistema descentralizado necessitaria se auto-reconfigurações de maneira a manter os recursos mapeados entre os participantes. Por se tratar de um modelo descentralizado, a tarefa de se auto-organizar acaba sendo uma tarefa distribuída entre os próprios participantes do sistema, que pode contar com um ou vários nós mantendo um mapeamento dos recursos e caminhos aos participantes.

5.3 Classificação das redes peer-to-peer

Os sistemas P2P podem ser classificados (Rezende, 2009) quanto à forma como os índices dos recursos disponível entre os participantes são armazenados, a forma com que as buscas a determinados recursos são realizadas entre os participantes, e uma classificação

híbrida que leva em consideração tanto as listas quanto a forma de busca. A seguir serão mostradas as principais características de cada classificação.

5.3.1 Classificação quanto aos índices

Leva em consideração a maneira como os índices são mantidos na rede. Um índice de maneira geral contém a descrição do recurso ou objeto, e as suas localizações em relação à disponibilidade dos participantes da rede:

Modelo Centralizado: é mantido um índice global com as informações referentes aos recursos compartilhados. Os clientes acessam diretamente os recursos que desejam. O Napster é um exemplo de aplicação deste tipo.

Modelo Descentralizado: não é mantido nenhum tipo de índice global, geralmente as buscas passam por todos os participantes. Um exemplo desta classificação é o Gnutella.

Modelo Hierárquico: utiliza um modelo com participantes especiais, chamados super nós, que possuem características especiais, executam tarefas como a manutenção de índices. Um exemplo típico de aplicação deste tipo é o Kazaa. Este modelo é entendido como um modelo intermediário entre o modelo centralizado e o descentralizado.

5.3.2 Classificação quanto ao tipo de busca

Nesta classificação é levado em conta o fluxo de uma busca por um recurso em uma rede P2P. Podemos ter as seguintes classificações quanto aos tipos de busca:

Serviço de Localização Centralizada (Centralized Service Location - CSL): a busca é realizada em servidores centrais de índices os quais fornecem o participante que possui o recurso compartilhado, como o caso do Napster;

Serviço de Localização por Inundação (Flooding-based Service Location - FSL): utiliza o reenvio da requisição entre os nós participantes, por exemplo, Gnutella. O termo inundação faz referência ao número de mensagens que são enviadas a rede até que o nó desejado seja encontrado;

Serviço de Localização baseado em DHT (Distributed Hash Table-based Service Location DHT): utiliza um mecanismo de busca semelhante às tabelas hash, porém, mantém

partes desta tabela em diversos nós que compõem a rede. Uma tabela *hash* possui a informação sobre o recurso vinculado a um identificador que corresponde ao participante que possui o recurso em questão formando um par [*id participante, recurso*], no modelo base em DHT esta tabela é compartilhada entre todos os participantes da rede. Um exemplo que segue este modelo é o Pastry (Rowstron & Druschel, 2001).

5.3.3 Classificação híbrida

Nesta classificação procura-se alinhar as formas de armazenamento dos índices com os mecanismos de buscas entres os participantes. Levando em conta estes dois parâmetros temos:

Modelo centralizado: corresponde ao modelo centralizado de armazenamento do índice (mantém um índice global) e possui mecanismo de busca igual ao modelo CSL, ou seja, a busca também é centralizada no servidor que mantém os índices. Esta é a forma mais simples de implementação geralmente utilizado quando se busca qualidade na obtenção do recurso e não na busca por ele. Esta abordagem será adotada para fins experimentais no modelo proposto neste trabalho, pelo fato dos recursos serem bem definidos e conhecidos pelo servidor, resolvendo assim o problema de gargalo durante a requisição de recursos por vários clientes.

Modelo descentralizado e estruturado: esta classificação adota o modelo descentralizado de armazenamento dos índices, e possui mecanismos de buscas baseados em DHT, ou seja, mantém tabelas do tipo [*id participante, recurso*] distribuído entre os participantes;

Modelo descentralizado e não estruturado: mantém os índices armazenados de forma descentralizada ou ainda de forma hierárquica (que contém os super nós), e faz uso do mecanismo de buscas FSL, que conforme a segunda classificação realiza buscas por inundação, repassando a requisição para os participantes subseqüentes até o participante desejado.

5.4 Aplicações peer-to-peer

Dentre as principais aplicações das redes P2P podemos destacar os seguintes exemplos:

Compartilhamento de arquivos. A popularização das redes P2P se deu principalmente pelos programas de compartilhamento de arquivos, dentre eles o Napster, Kazaa, Gnutella e o FreeNet (Clarke, Sandberg, Wiley, & Hong, 2001). Como discutido anteriormente, possuem diferentes tipos de busca e armazenamento dos recursos. Geralmente gratuitos estes softwares se tornaram rapidamente populares, assim os usuários mantinham compartilhados arquivos como de música (mp3), filmes, jogos e até mesmo softwares são compartilhados de maneira indevida.

Computação distribuída. Um exemplo bem conhecido da utilização dos conceitos de peer-to-peer para computação distribuída é o projeto SETI@home (Anderson, Cobb, Korpela, Lebofsky, & Werthimer, 2002). O projeto iniciado em 1997, e foi realizado pela Universidade de Berkley em conjunto com a NASA. Ele é constituído por uma coleção de projetos de pesquisas científicas voltadas para a descoberta de civilizações extraterrestres. Nele, computadores clientes utilizam um protetor de tela que carrega os dados do servidor do SETI@home e, então, processam e enviam os resultados de volta ao servidor quando possuem capacidade de processamento ociosa.

Sistemas colaborativos. Fornecem uma infra-estrutura de colaboração e comunicação direta entre computadores e usuários. Geralmente tais sistemas incluem chats, mensagens instantâneas e aplicações compartilhadas. Os usuários podem se organizar em grupos e iniciar determinadas tarefas em conjunto. Um exemplo de aplicação é o Groove (Microsoft Groove, 2007) que oferece espaços compartilhados para o trabalho colaborativo, incluindo ferramentas para interação de grupos, compartilhamento de conteúdo, comunicação e atividades em comum em tempo real.

5.5 Considerações finais

O capitulo traz uma visão geral das redes peer-to-peer, o objetivo dele não era se aprofundar nas técnicas e algoritmos empregados na arquitetura peer-to-peer, mas sim apresentar as principais características desta arquitetura de rede.

O capítulo ainda mostrou os diferentes tipos de buscas e modos de distribuição de índices, que podem ser unidos para compor ainda uma classificação híbrida.

Os conceitos discutidos neste capítulo serão incluídos no modelo SW-V de streaming de software no próximo capítulo.

Capítulo
6

Modelo SW-V de streaming de software

Este capítulo apresenta o modelo SW-V de streaming de software, objetivo deste trabalho, que venha a se adequar em ambientes onde as tradicionais soluções não seriam viáveis pelo alto custo do investimento em infra-estrutura adequada ou ainda o alto valor que as atuais soluções do mercado possuem. Além disso, o modelo proposto vem a combinar técnicas propondo melhorias sobre os atuais trabalhos relacionados, criando ao mesmo tempo uma arquitetura leve e robusta de virtualização de aplicação. Dentre as principais técnicas envolvidas nesta proposta estão: o seqüenciamento inteligente da aplicação, onde é possível detectar exatamente os recursos essenciais que a aplicação necessita durante a execução; um middleware de virtualização a ser executado em *user-mode*, ou seja, sem a necessidade de privilégios administrativos; e a distribuição dos pacotes da aplicação através de uma rede distribuída, onde cada computador também se torna um servidor pela aplicação de conceitos de redes peer-to-peer.

Todos os componentes envolvidos no modelo de streaming de software que este trabalho se propõe serão apresentados nas seções seguintes.

6.1 Introdução ao modelo SW-V

Como apresentado no capítulo 4 deste trabalho, existem soluções similares no mercado projetadas para realizar streaming de aplicação, mas de acordo com um levantamento recente (Huisman & Haverink, 2009), todos eles apresentam pontos fracos como compatibilidade com aplicações 64 bits, possibilidade serem utilizados no contexto da internet, em sua grande maioria necessitam que componentes serem instalados previamente nos clientes e ainda precisam ser executados em *kernel-mode* (privilégios especiais).

Em estudo relacionado (Zhang, Wang, & Hong, 2009), procurou-se associar as técnicas de distribuição peer-to-peer ao streaming de software, porém, o modelo apresentado neste trabalho apresenta algumas limitações similares às soluções já existentes, como por exemplo a necessidade de privilégios especiais para a execução cliente. Outro ponto falho no trabalho citado é a falta de um mecanismo que consiga detectar e separar os recursos que a aplicação irá requisitar primeiro, isso aumenta o tempo em que a aplicação irá demorar a iniciar. Por fim o trabalho apresenta uma arquitetura envolvendo dispositivos de armazenamento portáteis como pontos de partida da execução de uma aplicação, em nosso modelo nenhum hardware adicional é necessário.

O modelo proposto neste trabalho explora o sistema operacional hospedeiro ao máximo sem que o mesmo seja modificado (obedecendo a definição de virtualização de aplicação discutida na seção 4.1). De modo geral, os principais módulos envolvidos na proposta são: (1) módulo de criação da imagem da aplicação. (2) módulo de distribuição da imagem do software e (3) módulo de virtualização que corresponde à camada que irá abstrair a instalação do software no sistema operacional hospedeiro.

A figura 24 mostra os processos que um modelo de streaming de software deve percorrer:

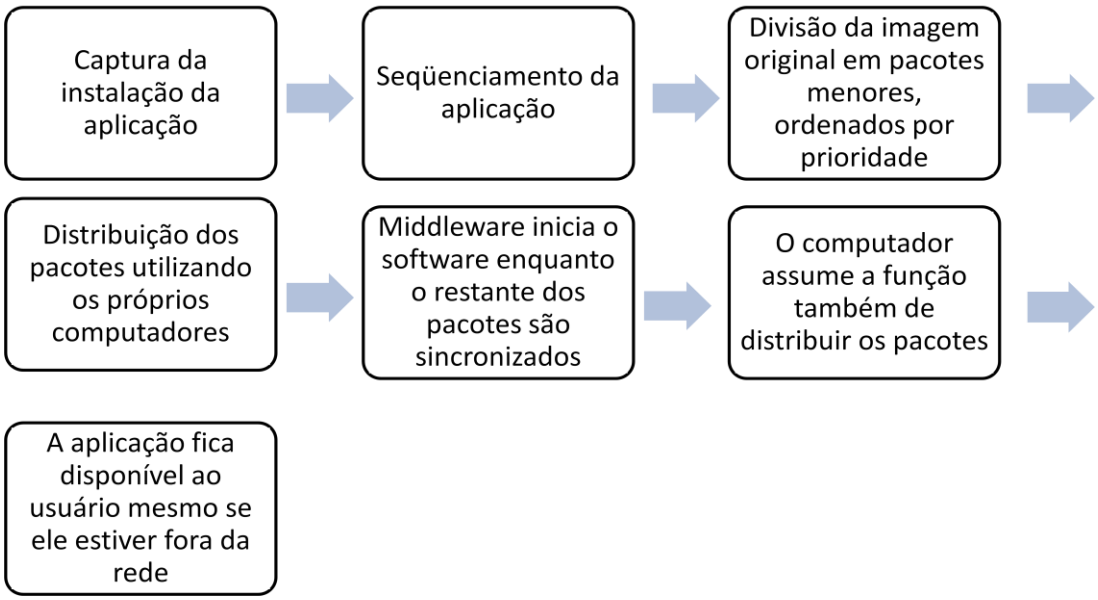


Figura 24 – Principais processos envolvidos em uma solução de streaming de software

A seguir serão apresentados de forma detalhadas estes processos e os demais módulos envolvidos na arquitetura SW-V.

6.2 Criação da imagem da aplicação

Uma imagem completa do software precisa ser montada para ser distribuída para os computadores. A execução da aplicação final no cliente depende diretamente deste módulo da arquitetura, pois caso tenha alguma falha, como uma biblioteca da aplicação não ter sido capturada corretamente, refletirá diretamente na experiência do usuário durante a utilização posterior do mesmo.

Para se criar a imagem de um software, é preciso saber exatamente o que uma aplicação utiliza, pois no momento de sua execução em um computador cliente, o middleware precisará reconhecer que o a aplicação precisa do recurso e irá direcionar a requisição do mesmo para o lugar correto onde o recurso está localizado.

Os principais recursos que uma aplicação utiliza modo geral são:

- ***Arquivos***: DLL, outras bibliotecas de funções, imagens e arquivos de configurações, outros executáveis, e etc.

- ***Registros***: Chaves de registros que geralmente são criadas durante a instalação do software. Algumas instalações podem ao invés de criar, modificar chaves já existentes, estas alterações devem ser capturadas também.

O processo de criação da imagem do software é divido em subprocessos como veremos a seguir.

6.2.1 Captura da instalação do software

No modelo proposto o mesmo middleware de virtualização responsável por interceptar as chamadas que a aplicação realiza durante sua execução é empregado também para capturar todas as alterações que um programa de instalação realiza. O instalador nada mais é do que um software que realiza basicamente procedimentos de cópia de novos arquivos, inclusão ou modificação de registros do sistema, eventual alteração de variáveis de ambiente e assim por diante. O middleware (como veremos na seção 6.5) possui um mecanismo que intercepta e

tratam as chamadas as API do sistema operacional. Portanto o processo de captura da instalação consiste na execução supervisionada do instalador do software em questão.

Neste modo de captura, ao invés do middleware agir direcionando as chamadas para as localizações reais dos recursos, ele age simplesmente documentando todos os passos realizados pelo instalador. Ao término da instalação, o resultado desta fase consiste em duas listas contendo todas as operações realizadas com arquivos e todas as operações realizadas com registros.

Exemplo de um trecho da lista de operações com arquivos:

```
C:\Program Files\Adobe\Reader 9.0\Reader             1KB    D
C:\Program Files\Adobe\Reader 9.0\Reader\A3DUtility.exe    252KB        A
C:\Program Files\Adobe\Reader 9.0\Reader\ACE.dll           786KB        A
```

6.2.2 Capturando os arquivos

Utilizando-se da lista de alterações realizadas no sistema de arquivos, é lançado um subprocesso que irá analisar esta lista e agrupar em um único diretório os arquivos que foram monitorados durante a instalação. Uma cópia de todos estes arquivos será realizada, com isso iremos gerar a pasta imagem do software. O caminho (ou *path*) dos arquivos serão tratados para que a pasta destino onde serão inseridos se torne a raiz deste novo caminho. Por exemplo, um arquivo que originalmente foi colocado em "c:\program files\Adobe\Reader" ao ser transferido para a pasta imagem, irá possuir a localização "c:\<*PastaImagemSoftware*>\ Program Files\Adobe\Reader". Assim eliminamos a necessidade futura do middleware ter que realizar uma busca por um arquivo específico quando for solicitado, já que mantendo a hierarquia de pasta, quando a aplicação requisitar o arquivo, basta substituir o drive "C:\" pelo endereço da pasta imagem que o arquivo será encontrado de forma direta.

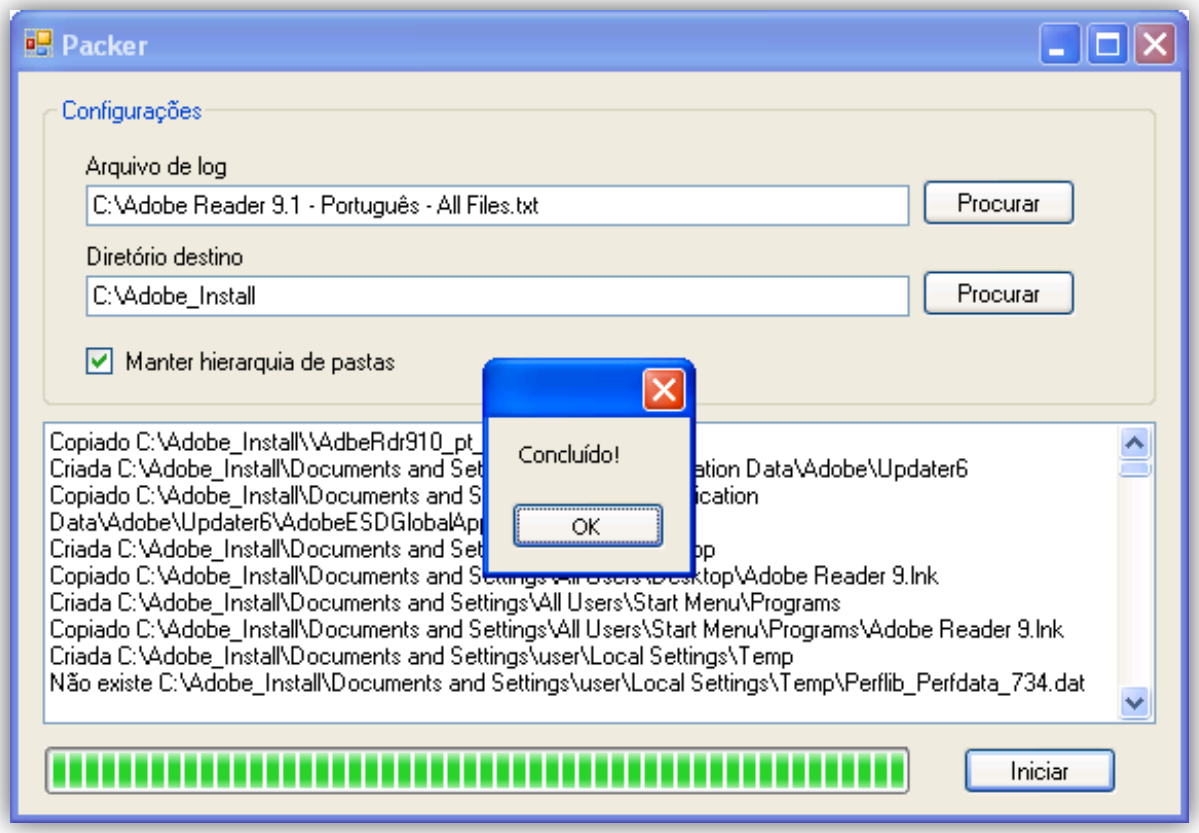


Figura 25 - Processo criado para analisar a lista de alterações no sistema de arquivos

Após o processamento da lista das modificações em arquivos, mostrado na figura 25, teremos uma pasta imagem do software, contendo todos os arquivos que foram incluídos ou modificados pelo instalador, conforme é mostrado na figura 26:

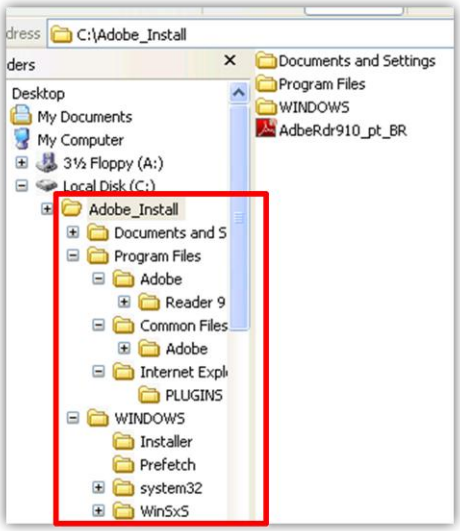


Figura 26 - Pasta imagem da aplicação gerada após o processamento da lista de alterações no sistema de arquivos durante a instalação

Tendo todos os arquivos já separados, é necessário realizar a mesma operação de tratamento com os registros que foram alterados ou incluídos no sistema.

6.2.3 Capturando as chaves de registros

Da mesma forma como foi realizado na separação dos arquivos que serão usados pela aplicação, é necessário criar uma estrutura que contenha os registros que foram incluídos ou modificados durante o processo de instalação do software. Este subprocesso portanto, analisa a lista de modificações em registros e gera uma estrutura intermediária que facilitará a operação de recuperação por parte de middleware quando determinado registro for solicitado.

A lista com as operações possui a seguinte estrutura:

```
[HKEY_CURRENT_USER\Software\Adobe\Acrobat Reader\9.0\AdobeViewer]
"TrustedMode"=dword:00000000

[HKEY_CURRENT_USER\Software\Adobe\Acrobat Reader\9.0\InstallPath]
@="C:\\Program Files\\Adobe\\Reader 9.0\\Reader"
```

Este formato necessita ser alterado para uma estrutura que facilite as buscas pelo middleware enquanto a aplicação estiver em execução. A busca em um arquivo continuo de texto utilizando comparação direta por string, resulta em um tempo maior que a aplicação levará para ser iniciada. Para suprir essa falha, diferente de todos dos modelos existentes estudados até aqui, o modelo proposto produz uma estrutura no formato XML que facilita a busca pelos registros. Utilizando este formato, o middleware não precisará percorrer o arquivo todo ou criar índices para encontrar determinada chave de registro. Basta informar o caminho completo no XML que resultará na chave desejada. A figura 27 mostra um trecho de como o arquivo de registros fica estruturado em formato XML:

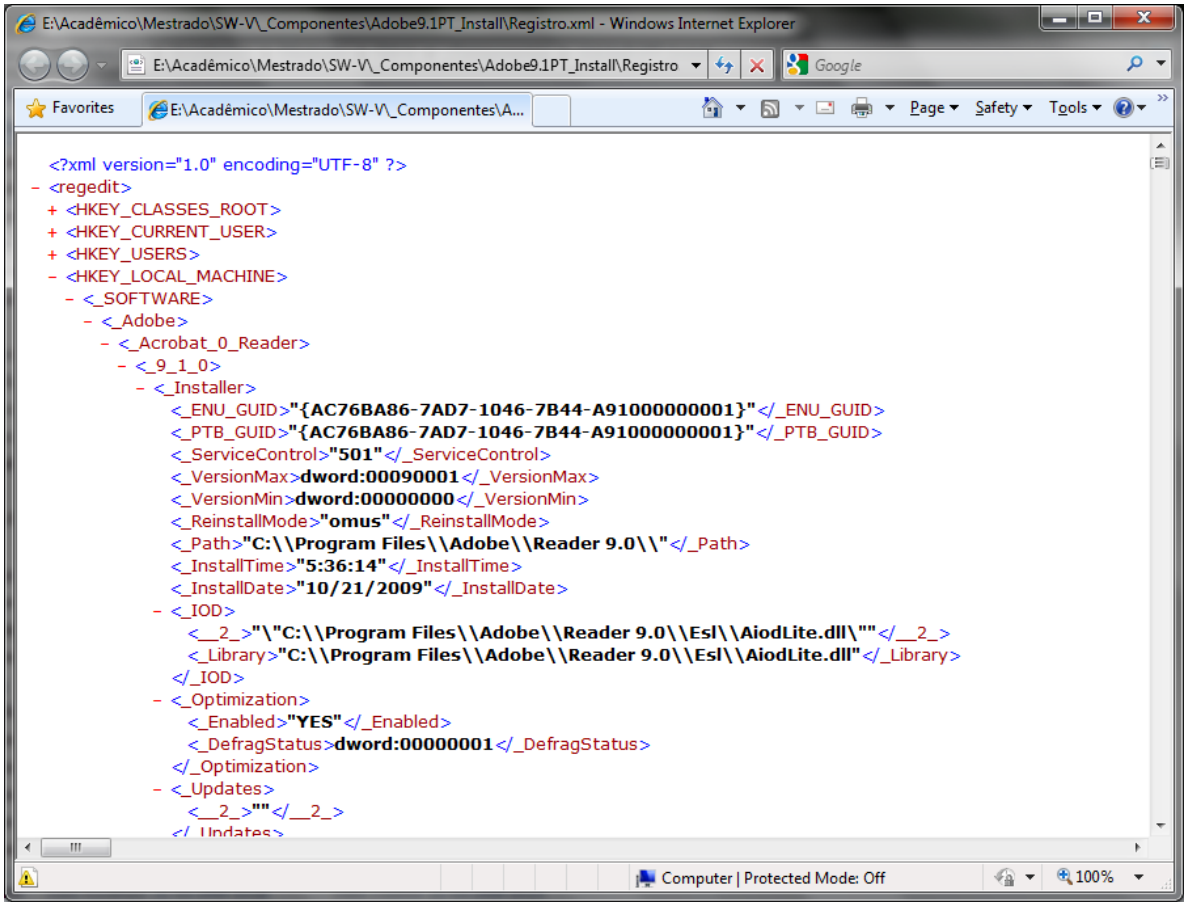


Figura 27 - Estrutura de um arquivo de registros já em formato XML. Promove a busca estruturada no arquivo

Algumas palavras do arquivo XML foram substituídas por uma simbologia. Os arquivos XML possuem regras para se definir nomes de atributos, nós e valores, enquanto as chaves de registros possuem inclusive caracteres especiais. Para resolver este problema, os caracteres que causam problemas no endereço da chave são substituídos por símbolos. Ao se fazer a procura por uma chave de registro no arquivo XML (que já contém os símbolos), o caminho é então transformado no padrão que o XML é formatado, ou seja, todos os caracteres especiais são substituídos pelos símbolos do mesmo dicionário utilizado pelo arquivo XML, assim o endereço final será o mesmo que foi utilizado durante a criação do arquivo XML. “Os caracteres até o momento identificados e simbolizados são: “. (ponto), @, %, &, {, }, /, *, +, \\, :, (e)”.

6.2.4 Imagem da instalação da aplicação

Ao término do processo de tratamento dos registros, teremos a imagem da aplicação que consiste dos arquivos (incluindo a lista dos arquivos), os registros no formato original e os registros em formato XML como mostra a figura 28Figura 28.

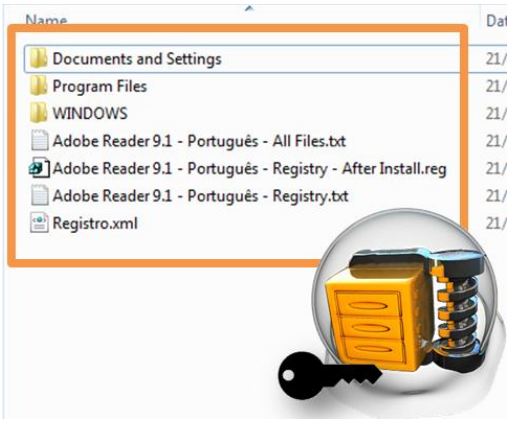


Figura 28 - Imagem da aplicação composta dos arquivos e registros que o instalador incluiu ou modificou.

Esta imagem será posteriormente compactada e protegida de alteração, permanecendo uma cópia fiel da instalação do software. Porém o processo da criação da imagem do software para ser distribuído no modelo proposto ainda passa por um processo que é outro diferencial: seqüenciamento assistido da aplicação.

6.3 Seqüenciamento assistido da aplicação

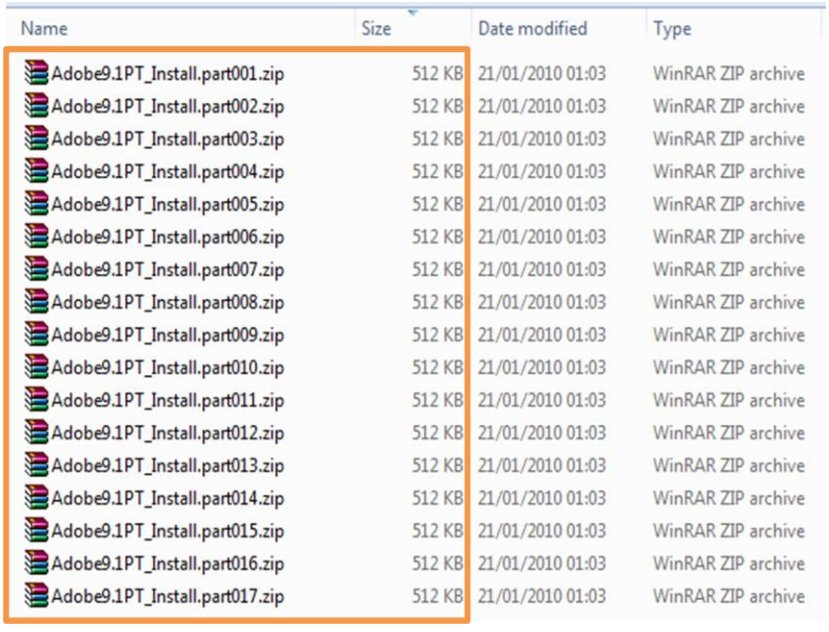
Tomando alguns conceitos já explorados pelo Microsoft App-V, que seria monitorar uma primeira execução do software, o modelo proposto procura incluir uma interação com o usuário que está fazendo a instalação para posterior compartilhamento da aplicação.

Neste momento do processo, já possuímos os recursos que o instalador da aplicação utilizou, mas como dito anteriormente, muitos recursos instalados não são utilizados necessariamente em um primeiro momento quando o software é executado, estes então podem ser considerados de prioridade menor, e os recursos essenciais para a execução da aplicação como recursos de maior prioridade e necessitam serem entregues primeiro. Um recurso dito como prioritário é o arquivo de registros gerados no processo anterior (arquivo XML).

Sendo assim, o modelo propõe a execução monitorada da aplicação, onde através de wizzard de instruções, o usuário é levado a executar as principais ações para o qual o software

foi desenvolvido. Por exemplo, caso o software em questão fosse um processador de textos, o usuário seria conduzido a executar ações básicas no software como troca de fonte, definição de parágrafos, inserção de uma imagem e assim por diante. Isto irá assegurar logo que a aplicação seja iniciada, o usuário consiga executar as funções básicas do software enquanto que o restante dos pacotes que compõe a imagem da aplicação é recebido de forma assíncrona. Um exemplo da diferença do tamanho instalado para o tamanho necessário para execução é o software Adobe Acrobat Reader 9.1 que instala aproximadamente 218 Mbytes entre arquivos e registros e utiliza um pouco mais de 44.1 Mbytes para ser executado e utilizado em suas funções básicas. Este seqüenciamento acelera a inicialização do aplicativo, pois ao invés de baixar toda a instalação, uma pequena parte já é o suficiente.

Para se obter os recursos mais utilizados, o middleware é utilizado novamente em modo estatístico. Ao ser inicializado ele irá começar a gerar listas semelhantes às geradas durante a instalação, com a diferença que irá conter apenas os recursos que foram referenciados durante este pequeno período de execução do software. Depois de finalizada a execução do software, estas listas serão novamente processadas (semelhante como aconteceu após a instalação) com a diferença que vários pacotes (arquivos de tamanho reduzido, geralmente em torno de 0,5 a 1 Mbyte(s)) serão produzidos em ordem de prioridade, ou seja, conforme a seqüência em que foram referenciados durante o seqüenciamento, conforme mostra a figura 29.



Name	Size	Date modified	Type
Adobe9.1PT_Install.part001.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part002.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part003.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part004.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part005.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part006.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part007.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part008.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part009.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part010.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part011.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part012.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part013.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part014.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part015.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part016.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive
Adobe9.1PT_Install.part017.zip	512 KB	21/01/2010 01:03	WinRAR ZIP archive

Figura 29 – Ao término do processo, a aplicação está seqüenciada e dividida em pacotes menores para ser enviadas para os clientes via streaming

Após o processo de criação de uma imagem de um software, um arquivo de configuração é gerado. Este arquivo irá conter informações úteis sobre o software, como por exemplo, a quantidade de pacotes que o processo gerou, quantos são os pacotes de prioridade alta (sem os quais a aplicação não poderá ser executada), versão do software, data em que foi gerado e assim por diante. Quando um cliente requisitar o software em questão, este arquivo de configuração irá servir como parâmetro dos passos a ser seguido pelo middleware para executar a aplicação.

6.4 Streaming da imagem da aplicação

Neste ponto temos a imagem da aplicação pronta para ser distribuída no servidor. Como comentado nas seções 6.2 e 6.3, o método de distribuição influenciará diretamente na experiência que o usuário terá na utilização do aplicativo. Em modelos anteriormente apresentados, o servidor de streaming é sempre centralizado, ou seja, todos os clientes precisam necessariamente pegar os pacotes do servidor. Pensando em um ambiente de um laboratório de informática de uma escola por exemplo, que segue um roteiro que envolve os alunos a utilizarem e iniciarem determinados programas ao mesmo tempo, esse modelo centralizado iria gerar gargalos nesta estrutura, pois geralmente os alunos iriam executar os mesmos programas ao mesmo tempo. Saindo do contexto de escola, podemos citar as empresas de telemarketing como exemplo também, sempre a cada troca de turno, os sistemas deverão ser acessados por um grande contingente de usuários em horários muito próximos.

O modelo SW-V propõe então uma abordagem distribuída, utilizando-se de conceitos de redes P2P, onde cada computador que já tenha copiado pelo menos um pacote, já poderá se tornar um servidor auxiliar para os demais, isto irá evitar que haja gargalos de requisições no servidor, fazendo com que inclusive os computadores iniciem as aplicações mais rapidamente como mostrado na figura 30.

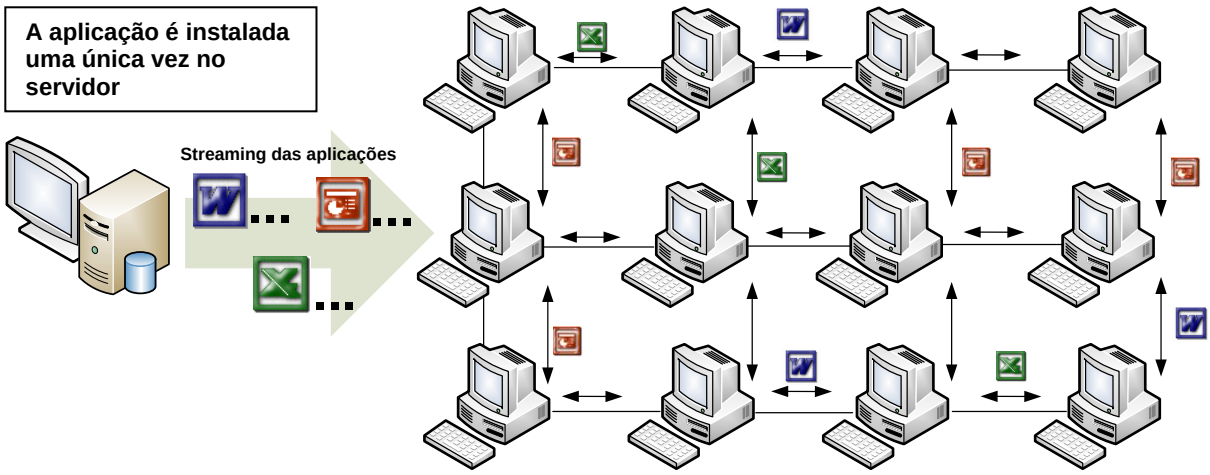


Figura 30 - Modelo distribuído de streaming de software utilizado conceitos de redes P2P, ao receber um pacote o computador se torna automaticamente em um servidor auxiliar

Para que este mecanismo funcione, o servidor mantém uma lista de todos os computadores da rede que já requisitaram o software (modelo centralizado de rede peer-to-peer). Assim quando um novo computador faz a requisição do software ao servidor ele envia o arquivo de configuração do software (discutido no final na seção 6.3) e a lista de outros computadores da rede com os pacotes que cada possui como mostrado na figura 30. Com estas informações, ele irá começar a requisitar aos outros computadores que forneçam os pacotes, balanceando os pedidos entre os computadores disponíveis. Caso tenha dois ou mais computadores prontos para atender, ele partirá para baixar pacotes de computadores diferentes ao mesmo tempo, reduzindo o tempo total que levará para a cópia dos arquivos de alta prioridade.

Após ter baixado os primeiros pacotes (prioritários) o computador irá avisar ao servidor que está pronto para também atender requisições, e em seguida, continuará requisitando os demais pacotes para os computadores que ainda estiverem disponíveis.

Todo este mecanismo de sincronização dos pacotes fica contido no próprio middleware de virtualização. Isto facilita, pois as únicas peças que o cliente irá precisar para obter um software neste modelo, é o middleware (que é auto-executável, não requer instalação) e o arquivo de configuração do software. Na próxima seção será tratado este importante componente, bem como o fluxo que ele irá percorrer durante uma solicitação.

6.5 Middleware de virtualização

A principal função de um middleware para virtualização de aplicação é interceptar chamadas de sistema (VDI works, 2008) (Russinovich, Solomon, & Ionescu, 2009), principalmente chamadas a operações no sistema de arquivos, funções de registros e funções específicas como a execução de processos adicionais. Em trabalhos relacionados (Yu, Guo, Nanda, Lam, & Chiueh, 2006) (Zhang, Wang, & Hong, 2009) (Alpern, Joshua, Bala, Frauenhofer, Mummert, & Pigott, 2005) este mecanismo de interceptação foi empregado sempre operando em *kernel mode* do sistema operacional, ou seja, todos eles precisavam de privilégios administrativos para serem executados e ainda, necessitavam da instalação de *drivers* ou a instalação de filtros no sistema de arquivos sistema operacional (Russinovich, Solomon, & Ionescu, 2009). O modelo proposto utiliza um mecanismo mais próximo da aplicação do usuário que é a injeção de DLL. Quando uma aplicação é executada ela pode lançar diversas *threads* de controle, onde sempre uma thread será dita a thread principal do programa. Durante a criação dessa thread principal (que a partir dela as outras serão criadas) todas as API que serão utilizadas do sistema operacional são referenciadas através de ponteiros. Esse mecanismo (*DLL injection*) funciona substituindo a função referenciada originalmente por uma função programada que fará o tratamento da chamada, assim sendo é possível sobrescrever algumas API do sistema operacional. A vantagem desse mecanismo é que ele executa no mesmo nível de privilégio do usuário, não requer instalação de nenhum componente prévio e ainda, e talvez a maior de todas as vantagens, a substituição da função original somente é reconhecida pelo programa em questão, todos os outros processos e aplicações que estiverem em execução ao mesmo tempo, irão fazer as chamadas às funções originais do sistema.

As outras abordagens citadas anteriormente, ainda utilizam filtros diretamente sobre o sistema de arquivos em uma camada em nível de *driver*. Devido a isso, todos os processos, incluindo as chamadas que o próprio sistema operacional faz, passa pelo middleware. Isto acarreta numa sobrecarga neste filtro, pois todas as chamadas de todos os processos precisarão ser analisadas para identificar se são chamadas realizadas pelo software virtualizado. A figura 31**Figura 31** mostra os diferentes níveis de privilégios de execução em um sistema operacional.

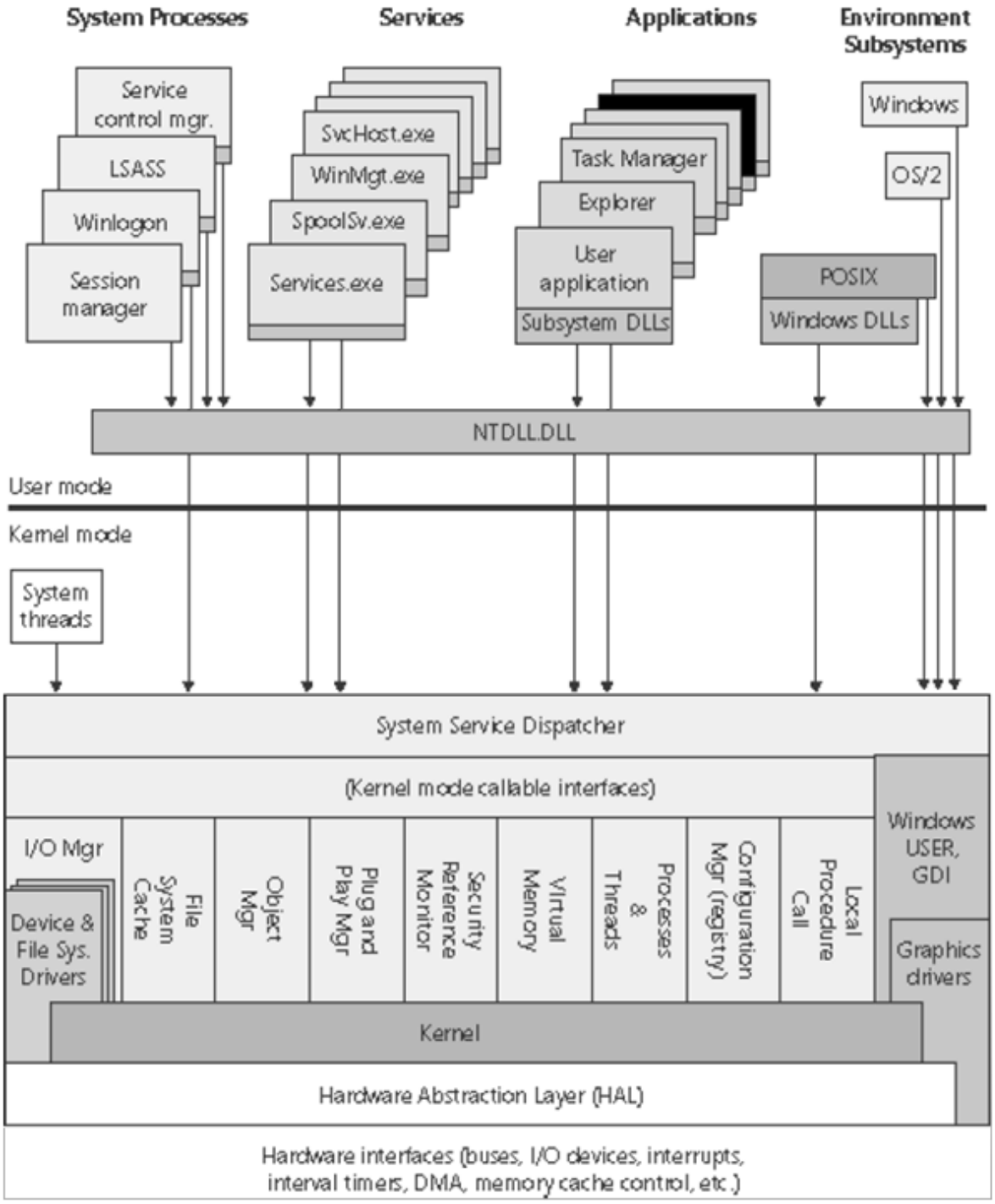


Figura 31 - Níveis de execução de um sistema operacional

Sendo a base do middleware a interceptação de API do sistema operacional, quando a aplicação chama determinada função da API do sistema hospedeiro, o middleware intercepta a chamada e redireciona para o ambiente virtual criado para a execução da aplicação. Neste ambiente alguns submódulos são necessários devido aos conceitos básicos da virtualização (Barham, Dragovic, Fraser, Hand, & Harris, 2003) ainda serem válidos no contexto da virtualização de aplicação, principalmente o conceito de uma aplicação não precisar ser

adaptada para ser executada virtualmente; e qualquer alteração feita pela aplicação virtualizada é imediatamente refletida apenas no ambiente virtual ao qual está restrita, não permitindo que o sistema hospedeiro seja modificado.

Os principais submódulos do middleware com suas principais características são descritos a seguir, o objetivo é mostrar as diferenças entre a proposta deste trabalho dos demais trabalhos relacionados e outras soluções comerciais já existentes.

6.5.1 Sistema de arquivos virtual

Uma das principais peças do ambiente virtual de execução é o sistema de arquivos virtual. Ele permite um ambiente isolado do sistema do hospedeiro. Diferente de outras abordagens, o mecanismo proposto não intercepta todas as chamadas, mas somente as operações que intencionam modificar algo no sistema hospedeiro, ou funções que procuram abrir arquivos específicos que podem estar armazenado no ambiente virtual de execução e não no caminho original que a aplicação acredita que ele esteja.

Quando uma aplicação é iniciada, inúmeros acessos a arquivos (para obter informações sobre eles apenas) são disparados, e em sua grande maioria arquivos que fazem parte do sistema hospedeiro e não da aplicação específica. Como esse grande número de chamadas é para arquivos nativos e não busca modificar o sistema hospedeiro, o middleware permite que essas instruções sejam executadas nativamente sem interceptação reduzindo o tempo de espera acumulado que essa aplicação terá para ser inicializada por completo. Um exemplo claro de alta quantidade de chamadas logo na inicialização da aplicação, é quando a mesma é desenvolvida sobre Frameworks, como o caso da Microsoft .Net Framework. Isto facilita o desenvolvimento e não tem impacto na virtualização, pois todas as novas versões dos sistemas operacionais Windows da Microsoft já vem com ela.

Em geral as operações de arquivos são realizadas sobre a API do sistema operacional, e principalmente passam pela função *CreateFile()* e *OpenFile()*. Quando uma aplicação precisa escrever em determinado arquivo, por exemplo, a função *CreateFile* é chamada com o parâmetro *write mode*. Se a aplicação deseja criar um arquivo novo, o middleware redireciona o caminho original que a aplicação deseja criar o arquivo para o espaço virtual; se a aplicação irá modificar um arquivo já existente no hospedeiro (detectado pelo parâmetro *OpenWrite*) neste momento é realizado uma cópia do arquivo original para a área do sistema de arquivos

virtual e ali são realizadas as alterações. A partir daí ao solicitar esse arquivo novamente, utilizando a função *OpenFile()* por exemplo , o middleware irá sempre utilizar a versão do espaço virtualizado. Este método foi chamado de *copy-on-change* em nossa proposta.

As outras API que realizam operações com arquivos, em geral, utilizam o *handle* (ponteiro) retornado pela função *CreateFile()*, ou seja, na maior parte do tempo, essa função é sempre chamada a cada operação com arquivos.

6.5.2 Espaço virtual de registros

Um método similar ao de arquivos é usado para montar um ambiente de registros virtual para compor o ambiente de execução virtual. Os registros são comumente usados para armazenar informações da instalação e outras configurações necessárias para execução do software. Mas a aplicação não usa somente seus próprios registros, mas constantemente a aplicação consulta informações sobre o sistema operacional e informações do usuário também. Quando a aplicação consulta registros, o espaço virtual é consultado, e caso o registro não seja encontrado o sistema hospedeiro é então consultado.

O espaço de registros virtual é similar ao mecanismo descrito em (Yu, Guo, Nanda, Lam, & Chiueh, 2006), mas com uma diferença para melhorar a busca no arquivo. Todos os registros capturados durante a instalação do software é transformado para o formato XML[1]. Desta maneira a busca pelo registro resume-se em especificar o caminho completo da chave, como em uma árvore binária. Isto reduz o tempo que o middleware levaria para procurar uma chave em um arquivo texto comum, onde todas as linhas precisariam ser testadas através de comparações se strings.

Quando uma aplicação consulta um registro, o caminho é então transformado no formato que o XML é estruturado, se o registro for encontrado todos os valores que a chave contém serão transferidos para uma área temporária em memória no registro do sistema operacional. Em geral, essa área temporária é contida em "*HKEY_CURRENT_USER\<application_name>\"*. Isto é necessário, pois todas as operações com registros utilizam ponteiros para as estruturas reais, que são retornadas pela API *RegOpenKeyExW()*. A aplicação passa o caminho completo do registro para esta função que verifica a existência e retorna um *handle* (ponteiro a uma estrutura protegida), após isso, todas as outras funções que

[1] XML (*Extensible Markup Language*) is a set of rules for encoding several data types electronically.

tratam registros utilizarão este *handle*. Utilizando um espaço temporário, porém real, no sistema operacional hospedeiro, não é necessário criar uma estrutura de dados especial para armazenar todas as chamadas a função *RegOpenKeyExW* () que foram realizadas, porque o ponteiro retornado é real, mesmo que apontando para uma área temporária.

Outras funções também precisaram ser monitoradas dentre elas *RegOpenKeyEx(), RegCreateKey(), RegCloseKey(), RegConnectRegistry(), RegFlushKey(), RegDeleteKey(), RegDeleteValue(), RegEnumKey(), RegEnumValue(), RegSetValueEx(), RegSetValueEx(), RegSetValueEx(), RegSetValueEx(), RegQueryValueE()* e *RegQueryValueEx().*

6.5.3 Monitor de criação de subprocessos

Algumas aplicações durante sua execução podem executar outros processos. Estes subprocessos precisam ser virtualizados também. Assim que a aplicação inicia outra aplicação ou processo, a API *CreateProcess()* é chamada e interceptada pelo middleware, se a aplicação alvo é uma aplicação nativa do sistema hospedeiro, a execução ocorre normalmente, porém se a aplicação é parte do software virtualizado, uma nova instancia do middleware é executado e ficará responsável por monitorar este novo processo criado.

Depois disto a execução do novo processo é contida também em um ambiente de execução virtual tal como a aplicação principal. Isto assegura que todos os recursos necessários serão também providos ao novo processo e todas as alterações que este vier a fazer será armazenado em um ambiente fora do sistema hospedeiro.

6.5.4 Interceptação de variáveis de ambiente

Durante a instalação do software as alterações feitas nas variáveis de ambiente são interceptadas também. Isto é alcançado fazendo a diferença entre o antes e o depois que o instalador do software é executado. Qualquer alteração é armazenada e quando a aplicação for ser executada no ambiente virtual, as variáveis de ambiente do sistema hospedeiro são atualizadas com as informações capturadas. Esta modificação não é persistente, ou seja, fica apenas em memória, não caracterizando uma modificação definitiva do sistema hospedeiro. O tratamento de variáveis de ambiente não é relatada nos trabalhos citados anteriormente.

6.5.5 Manifestos

Os manifestos são arquivos XML que descrevem o *assembly* que é utilizada pelo software (Russinovich, Solomon, & Ionescu, 2009). Este arquivo é usado para determinar a versão correta da biblioteca usada pela aplicação, bem como os dados da compilação dela. Na prática, todas as bibliotecas descritas neste arquivo precisam ser a mesma que será disponibilizada para a aplicação. O problema principal quando uma aplicação utiliza manifestos, é que ela não consegue ser iniciada se o arquivo de manifesto estiver fora de uma pasta específica, dentro da pasta do Windows. Para resolver o problema, uma área protegida nesta pasta foi criada. Todos os manifestos de uma aplicação são inseridos antes da execução dele e apagados após a aplicação terminar sua execução.

6.5.6 Utilização no modo off-line

Após a primeira vez que a aplicação é executada todos os seus recursos ficam em forma de cache no computador, inclusive um atalho é criado na área de trabalho do usuário. Caso o usuário execute essa aplicação fora do ambiente de rede, ou seja, desconectado do servidor e demais computadores, o middleware irá reconhecer essa desconexão e irá imediatamente trabalhar de forma off-line utilizando os arquivos do cache da aplicação.

Assim que detectado a volta da conexão de rede, o middleware deverá se conectar novamente ao servidor verificando se houve alterações nos pacotes da aplicação. Caso alguma alteração ou atualização seja constatada, o middleware irá novamente entrar em processo de streaming atualizando seus recursos. Imediatamente um aviso é dado ao usuário, mostrando que uma nova versão do software foi disponibilizada, que basta ele executar novamente a aplicação que já estará utilizando a versão mais nova.

6.6 Considerações finais

Neste capítulo foram apresentados os processos envolvidos no modelo SW-V, a seguir serão apresentados o protótipo realizado em cima dos conceitos do modelo proposto.

Capítulo
7

Protótipo

O objetivo deste trabalho é a proposta de um modelo de streaming de software que atenda as expectativas e demandas futuras como discutido no capitulo inicial. No entanto, alguns módulos foram desenvolvidos a fim de fundamentar a proposta. Atualmente, pequenos módulos em separado foram utilizados para testes, e futuramente, como será discutido na conclusão deste trabalho, uma solução completa é prevista. Neste contexto, alguns estudos de casos foram realizados nos módulos individuais, onde foi possível chegar a algumas comprovações.

7.1 Infraestrutura

Para os testes do protótipo foram utilizados computadores com processador Intel Core 2 Duo 2.0 Ghz com uma instalação limpa do Windows 7 Ultimate Edition 64 bits, que são as configurações comumente utilizada por computadores adquiridos atualmente. No momento que as aplicações eram testadas apenas os processos essenciais para o sistema operacional permaneciam em execução, evitando assim que pequenos gargalos fossem gerados por causa de outros processos de programas concorrentes.

Para os testes de transferência de arquivos entre os computadores, uma infraestrutura de computadores foi montada utilizando um switch comum padrão 10/100mbps, limitando assim a velocidade da rede ao padrão mais encontrado no mercado e nas empresas.

7.2 Codificação

O projeto do middleware foi codificado utilizando a linguagem C para a biblioteca que instala e intercepta as chamadas as API do sistema operacional hospedeiro, desta maneira tínhamos acesso a várias funções que linguagens de alto nível não teriam. Sobre essa camada

que intercepta os chamadas adicionamos uma interface onde parte dos procedimentos poderiam ser desenvolvidas em linguagem de mais alto nível, como por exemplo, a linguagem C# que executa sobre a Microsoft .Net Framework.

Os subprojetos tratamento de arquivos e registros coletados pelo middleware foram desenvolvidos também sobre a linguagem C# e .Net Framework devido a agilidade que essa linguagem oferece, e principalmente, porque não seria necessário medir o tempo que os arquivos ou registros eram processados, e que poderiam sofre um pequeno atraso devido ao uso da .Net Framework.

7.3 Módulo de criação da imagem da aplicação

Uma instalação comum do Adobe Acrobat Reader 9.1 capturada envolveu em torno de 218.12 Mbytes em novos arquivos e modificação e inclusão de registros. Usando um algoritmo de compressão comum como o Zip, o tamanho da imagem (completo) ficou em torno de 98.2 Mbytes.

Associando o método de seqüenciamento, a imagem básica contendo os recursos essenciais e de alta prioridade para a aplicação fica em torno de 44.1 Mbytes, utilizando o mesmo método de compressão, o pacote essencial cai para 22.2 Mbytes, que corresponde a 89.82% do pacote original.

7.4 Streaming da imagem da aplicação

Para que o usuário tenha o software completo mesmo estando sem conectividade com a rede, é necessário que toda a imagem da aplicação seja sincronizada. Como dito anteriormente, em primeiro plano somente os recursos essenciais são copiados e depois o restante da imagem. A imagem original da aplicação utilizado na fase anterior possui 98.2 Mbytes (comprimido). Usando um compartilhamento comum de rede o tempo para a sincronização foi de 9 a 10 segundos. Usando a abordagem P2P, somente o pacote essencial foi entregue primeiro pelo servidor e tomou em torno de 2.4 segundos para ser transferido, este é o tempo médio que a aplicação demorará a começar a ser iniciada. O restante da imagem foi dividida em pacotes menores de 512 Kbytes. Com mais um participante na rede (atuando como um servidor da imagem também), o tempo da sincronização do restante da imagem (em torno de 76.0 Mbytes) ficou em torno de 5 a 6 segundos. Em termos numéricos,

parece não ser significante, mas o maior benefício além da redução do tempo foi a redução do *overhead* do servidor, pois o servidor não precisou ser utilizado durante todo o processo de streaming, sendo que com vários clientes ele levaria um tempo maior para atender a todos eles. Ao utilizar esse mecanismo o servidor sempre estava presente para novos clientes requisitando o software.

7.5 Middleware de virtualização

Alguns dados foram obtidos comparando uma execução comum (através da instalação local da máquina) com a execução virtualizada. Os números não consideram o tempo de streaming dos pacotes. Usando a mesma aplicação capturada e distribuída nos tópicos anteriores (Adobe Acrobat 9.1), uma execução a partir de uma instalação local ocorre em torno de 0.8 segundos, enquanto que o tempo para a execução virtualizada com o middleware fazendo as devidas interceptações ficou em torno de 1.7 segundos. Este estudo preliminar deste módulo foi motivador comparando com outros sistemas relacionados (Huisman & Haverink, 2009), e estudos relacionados mostram uma diferença maior entre a execução do sistema virtualizado e a execução do sistema instalado.

7.6 Comparação do protótipo

A tabela Tabela 1 mostra as diferenças do modelo SW-V que podemos pontuar em relação a modelos e soluções já existentes no mercado.

Característica	*APP-V (MS)*	*ThinApp (VMWare)*	*XenAPP (Citrix)*	***SW-V***
Instalação módulo servidor	X	X	X	X
Não necessita de módulo cliente				X
Executável em user-mode		X		X
Compatibilidade x64				X
Gerenciamento de licenças	X		X	
Não necessita de um computador limpo para captura do snapshot da aplicação				X
A aplicação pode ser salva em um único executável		X		

Cache em pen-drive		X		X
Processamento exclusivo no cliente				X
Modelo descentralizado (P2P, por exemplo)				X
Web		X (deficiência)		X
Custo	-	$5000 (50 users) AppSuite $1000 Client Suite $10	$380 (5 users)	-

Tabela 1- Comparação do protótipo com modelos atuais

7.7 Considerações finais

O desenvolvimento do modelo, mesmo que não completo, foi necessário para provar que os conceitos propostos pelo trabalho são aplicáveis no contexto real dos computadores. Mesmo que não automaticamente acoplados, os módulos prototipados mostraram resultados satisfatórios comparados a trabalhados já existentes. Uma tabela mais ampla, com mais itens de comparação, poderá ser feita futuramente com base na solução completamente desenvolvida.

No próximo e último capítulo deste trabalho serão apresentadas as conclusões finais, e discutidos limitações e estudos futuros que trabalho possibilita.

Capítulo
8

Considerações finais

Neste capítulo serão apresentadas as considerações finais a partir do desenvolvimento deste trabalho, incluindo uma seção de conclusões mostrando como os objetivos inicialmente apresentados foram atingidos, bem como as conclusões que podemos chegar a partir dos capítulos anteriormente apresentados. Em seguida, uma seção de trabalhos futuro apresenta algumas oportunidades continuidade e evolução do modelo SW-V proposto neste trabalho.

8.1 Conclusões

O Modelo SW-V apresentou melhorias com relação às soluções existentes, propôs mecanismos que permitem a rápida execução das aplicações e criou uma nova forma de distribuição das imagens das aplicações utilizando a própria infra-estrutura disponível, tornando a solução mais escalável que as existentes no mercado.

Este trabalho mostrou os processos que um modelo de virtualização de aplicação com streaming de software deve adotar para se adequar as atuais e futuras demandas por essa tecnologia. Todos os módulos com suas peculiaridades foram discutidos, desde a criação da imagem, com técnicas para aperfeiçoar e reduzir os pacotes essenciais enviados ao computador cliente, passando pelo mecanismo de distribuição da imagem em que através da adoção de técnicas peer-to-peer distribui a responsabilidade do servidor entre os participantes da rede, chegando finalmente ao middleware de virtualização que é o coração da solução, onde todas as interceptações e desvios principais foram discutidos.

Outro objetivo que se buscou alcançar durante todo o trabalho foi comparar as técnicas do modelo proposto com os trabalhos relacionados e soluções de mercado já existentes. Como uma dessas diferenças marcantes e fortes do modelo SW-V é a adoção do mecanismo de seqüenciamento do software, que reduziu o tempo que o usuário espera para ter sua aplicação

executando. Isso permite que a diferença da execução do software de maneira instalada e de maneira virtualizada seja muito pequena ou quase imperceptível. Diferente do mecanismo adotado por outras soluções, o mecanismo do modelo proposto é assistido pelo usuário que é orientado a realizar as principais ações na aplicação, garantido assim que as funções básicas do software estarão disponíveis imediatamente na execução das mesmas. Outro problema identificado nas demais soluções existentes e trabalhos relacionados foi a incompatibilidade com sistemas operacionais 64 bits muito utilizados vistos que a maioria dos computadores já estão sendo montados com 4Gb de memória RAM.

Atendendo a outro objetivo proposto, o modelo SW-V utiliza a própria infra-estrutura de computadores disponível para dinamizar a forma que os novos participantes obtêm a imagem do software. Isso torna possível a adoção do modelo em organizações de menor porte, como pequenas empresas e escolas, onde geralmente nem servidores (computadores preparados para essa função) existem. Basta que qualquer máquina da rede seja eleita como a estação centralizadora das informações dos software disponíveis e todos os demais computadores atuarão como co-servidores. Junto com a redução de custos de infra-estrutura obtém-se também a redução de custos de licenciamento de sistemas operacionais específicos para servidores.

A arquitetura do middleware utiliza um mecanismo simples para interceptação que é a injeção de DLL, que não precisa de privilégios administrativos ou de sistema para funcionar. Isto elimina por completo a necessidade de instalação de agentes nos computadores clientes como é constatado nas demais aplicações de virtualização de aplicação que utilizam o streaming também. Além disso, em vários trabalhos relacionados, diversos componentes são empregados para se obter o software e executá-lo de forma virtualizada. Como visto anteriormente, o SW-V utiliza apenas de um componente no cliente responsável por todas as operações, que é o middleware de virtualização.

O SW-V se mostrou leve, seu middleware de virtualização precisa de poucos Kbytes e no computador cliente, a mesma quantidade em memória. Toda a execução do software consome exclusivamente o processamento da máquina local, diferentemente, por exemplo, da solução App-V, onde o servidor precisa acompanhar o processo de virtualização junto com o cliente monitorando o streaming das operações, conseqüentemente o servidor é ocupado com parte do processamento da aplicação no cliente.

Este modelo ainda pode ser adotado no contexto web, pois como visto, a transferência dos pacotes que compõe a imagem da aplicação é realizada através de um mecanismo que emprega técnicas peer-to-peer já adotas no ambiente da internet, possibilitando que vários pacotes sejam sincronizados ao mesmo tempo de fontes diferentes, evitando que a banda de um servidor no modelo centralizado venha a ser motivo de gargalo e demora no sistema. Ao contrário do modelo centralizado, a disponibilidade destes pacotes tende a aumentar a cada vez que há a entrada de um novo participante, visto que no modelo adotado ao mesmo tempo em que solicitam recursos eles se tornam também servidores dos mesmos.

Por fim outra contribuição deste trabalho foi uma revisão bibliográfica atualizada sobre técnicas e conceitos de virtualização, bem como um histórico do uso desta tecnologia.

8.2 Limitações do modelo

Durante a concepção do modelo SW-V foram encontradas algumas limitações. Dentre elas estão:

- ***Licenciamento de software***. Hoje o principal mecanismo de licenciamento de software é a aquisição de uma licença para a instalação do software. Como o modelo SW-V é capaz de capturar todas as modificações feitas pelo software, uma vez ativado o software durante a instalação, essa ativação é transferida para os computadores clientes, como acontece nas soluções de boot remoto. Para diminuir a possibilidade de cópias indevidas do cache armazenado nas estações foi empregado um mecanismo de criptografia dos pacotes da aplicação, com isso somente o middleware conseguirá utilizar esses pacotes. Porém a melhor maneira de resolver este problema é a adoção do licenciamento por uso, e não por instalação, aderindo assim a tendência do software como serviço, porém isso depende da adoção por parte da indústria de software.

- ***Aplicações com servidoras e serviços***. Aplicações como banco de dados e que são instaladas como serviços tendem a não obedecer a execução automática realizada durante o boot do sistema operacional, bem como a terem um comportamento inadequado. Quando uma aplicação caracterizada como serviço é instalada, o sistema operacional é configurado para executar de forma automática esse processo. O middleware atual não possui um mecanismo de integração que

possibilite o sistema operacional executar automaticamente aplicações virtualizadas.

- ***Atualizações automáticas.*** Atualizações dos software que foram virtualizados, precisam ser supervisionados pelo usuário, que deverá abrir a imagem, realizar as devidas atualizações, fechar a imagem novamente e realizar novo seqüenciamento do software. Não há um mecanismo automático que consiga sem a intervenção do usuário realizar as atualizações.

8.3 Trabalhos futuros e oportunidades de estudo

Como percebido durante o trabalho, e discutido nas atuais limitações do modelo SW-V, existem várias oportunidades de trabalhos futuros. Estes trabalhos e estudos poderiam envolver:

A) O desenvolvimento de uma solução completa integrando todos os módulos da arquitetura proposta. Sabe-se que o modelo é aplicável pelo desenvolvimento de pequenos módulos experimentais, mas devido ao curto período de tempo, não foi possível elaborar o modelo e ao mesmo tempo desenvolvê-lo por completo. Isto seria uma ótima oportunidade para se propor mecanismos adicionais que deverão ser identificados pondo o modelo em prática. Dentro deste plano de desenvolvimento, seria interessante um trabalho de pesquisa testando o uso desta solução em laboratórios de informática nas escolas. Até a conclusão deste trabalho, não foi possível encontrar nenhuma experimentação de virtualização de aplicação em escolas, o que seria o ambiente perfeito devido à enorme quantidade de softwares educacionais utilizados pelos professores, e que hoje é necessário a instalação de máquina por máquina.

B) Outro trabalho futuro poderia ser a resolução de algumas atuais limitações que a princípio seriam simples de se resolver que é a integração da aplicação com o Shell do sistema operacional e software que utilizam plugins. Tomando por exemplo o WinZip, quando instalado, disponibiliza em vários menus de contexto opções de compactação dos arquivos. Devido ao fato da aplicação ser instalada em outro computador e posteriormente ser transferida para outro computador automaticamente, essa integração não foi possível neste momento. Outra melhoria

relacionada seria a criação dos ícones de todas as aplicações acessíveis pelo usuário e aplicações que são instaladas como processos. Atualmente o modelo SW-V não possui mecanismo para identificar e diferenciar os aplicativos diretamente executados pelo usuário, dos programas complementares que funcionam como parte da solução e que são utilizados de forma automática pelo software. A resolução destes problemas estaria numa identificação inteligente dos registros responsáveis por estes mecanismos durante a instalação. Ao ser virtualizada traria a opção do usuário em replicar essas configurações no computador.

C) Outro ponto de estudo seria a compatibilização com diferentes versões de sistema operacional. Mesmo dentro do mesmo fabricante temos várias versões que requerem adaptações em software (como o Windows XP, Windows Vista e Windows 7). É interessante criar um mecanismo, que ao ser instalado o software ele possa capturar as diferenças entre a instalação nestas versões, e de acordo com o SO instalado no cliente, transferir os pacotes corretos. Isto pode ser feito interceptando a função que diz qual é o sistema operacional da máquina durante a instalação.

D) Outro trabalho futuro interessantíssimo seria a interoperabilidade com diferentes tipos de sistemas operacionais através de emuladores. Já é possível que uma aplicação Windows seja executada em sistemas Linux através de camadas de compatibilidade como o Wine e Mono. Fazendo com que o middleware de virtualização seja também executável em plataforma Linux é possível ter máquinas Windows e Linux usando os mesmos softwares e compartilhando a mesma rede e os mesmos recursos.

Referências Bibliográficas

Adams, K., & Agesen, O. (2006). A Comparison of Software and Hardware Techniques for x86 Virtualization.

Alpern, B., Joshua, A., Bala, V., Frauenhofer, T., Mummert, T., & Pigott, M. (2005). PDS: A Virtual Execution Environment for Software Deployment. *Proceedings of the First ACM/ USENIX International Conference on Virtual Execution Environments, March, 2005.*

Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., & Werthimer, D. (2002). Seti@home: an experiment in public-resource computing. *Commun. ACM* , v.45, n.11, p.56–61.

Androutsellis-Theotokis, S., & Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys (CSUR)* , v.36, n.4, p.335–371.

Barham, P., Dragovic, B., Fraser, K., Hand, S., & Harris, T. (2003). Xen and the Art of Virtualization.

Citrix Inc. (2006). Understanding Full Virtualization, Paravirtualization, and Hardware Assist.

Citrix Systems, Inc. (Julho de 2006). Application Virtualization -- Perceptions And Prospects.

Citrix XenApp. (2010). *Acessado em Agosto de 2009 em http://www.citrix.com.br/products/xenapp.php* .

Clarke, I., Sandberg, O., Wiley, B., & Hong, T. W. (2001). Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science* , v.2009, p.46.

Creasy, R. J. (1981). The Origin of the VM/370 Time-Sharing System. *IBM Journal of Research and Development* , pp. vol. 25, no. 5, p. 483.

Dike, J. (2000). A user-mode port of the Linux kernel. *In Proceedings of the 4th Annual Linux Showcase & Conference, Atlanta - USA* .

EIN-DOR, F. (1985). Grosch's Law Re-Revisited: CPU Power and The Cost of Computation. p. 142-151.

Fisher-Ogden, J. (s.d.). Hardware Support for Efficient Virtualization. *University of California* .

Goldberg, R. (1973). *Architecture of virtual machines AFIPS National Computer Conference* .

Goldberg, R., & Mager, P. (1979). Virtual machine technology: A bridge from large mainframes to networks of small computers. *IEEE Proceedings Compcon* , p. 210–213.

Golden, B., & Scheffy, C. (2008). *Virtualization for dummies (Sun and AMD special Edition).* Indianapolis, Indiana: Wiley Publishing, Inc.

Hu, T. H. (2005). Restructuring Peer-to-Peer Networks. *Tese (Doutorado), New South Wales* .

Huisman, S., & Haverink, M. (2009). Application Virtualization Comparison Chart September 2009. *VirtualFuture.info September 1s 2009* .

Intel. (2008). Choosing the Right Hardware for Server Virtualization.

Kamp, P., & Watson, R. N. (2000). Jails: Confining the omnipotent root. *In Proceedings of the Second International System Administration and Networking Conference (SANE)* .

Laureano, M. (2006). *Máquinas Virtuais e Emuladores: Conceitos, Técnicas e Aplicações.* São Paulo: Novatec.

Liu, H., Li, W., Wang, X., Gong, Y., & Luo, T. (2006). A Service-Oriented Virtual Machine for Grid Applications.

Microsoft Groove, C. (2007). *Acessado em Novembro de 2009 em http://www.microsoft.com/brasil/2007office/programs/groove/overview.mspx* .

Microsoft Softgrid. (2007). Microsoft SoftGrid application virtualization for Systems. *Acessado em Dezembro de 2009 em http://www.microsoft.com/systemcenter/softgrid/evaluation/delivery.mspx* .

Microsoft-b, C. (2007). Microsoft Completes Acquisition of Softricity . *Microsoft Presspas - Acessado em Maio de 2009 em http://www.microsoft.com/presspass/press/2006/jul06/07-17SoftricityPR.mspx* .

Popek, G., & Goldberg, R. (1974). Formal Requirements for Virtualizable Third Generation Architectures. *Comunications of the ACM* , pp. Volume 17, number 7, p.412-421.

Rezende, E. (2009). Modelo Estrutural para Compartilhamento de Arquivos Peer-to-Peer. M*estrado,* IBILCE/UNESP.

Robin, J. S., & Irvine, C. E. (14–17 de August de 2000). Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. *In USENIX, editor, Proceedings of the Ninth USENIX Security Symposium* , p. 275.

Rose, R (2004). Survey of System Virtualization Techniques.

Rosenblum, M. (2004). The reincarnation of virtual machines. Queue Focus. *ACM Press* , p. 34–40.

Rowstron, A., & Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. . *Lecture Notes in Computer Science* , v.2218, p.329.

Russinovich, M., Solomon, A. S., & Ionescu, A. (2009). *Windows® Internals: Incluing Windows Server 2008 and Windows Vista.* Redmond (June 17, 2009).: Microsoft Press, 5 edition .

Schollmeier, R. (2002). A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. *Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE* .

Singh, A. (s.d.). *An Introduction to Virtualization*. Acesso em 18 de 11 de 2008, disponível em http://www.kernelthread.com/publications/virtualization/

SMITH, J., & NAIR, J. (2005). The architecture of Virtual Machines. *University of Wisconsin-Madison: IEEE Computer Society* , p. 32.

Sugerman, J., Venkitachalam, G., & Lim, B. H. (2001). Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. *In Proceedings of the 2001 USENIX Annual Technical Conference* , p. 1-14.

Sun Inc. (2008). The VirtualBox architecture.

Ung, D., & Cifuentes, C. (2006). Dynamic re-engineering of binary code with run-time feedbacks. *Science of Computer Programming* , p. 189–204.

VDI works. (2008). How to build an Application Virtualization Framework. *Acessado em Julho de 2009 em http://vdiworks.com/wp/?p=15* .

VMWare Inc, 2008. (s.d.). *Virtualization Basics*. *Acessado em Julho de 2009 em*: http://www.vmware.com/overview/history.html

VMWare ThinApp. (2010). *Acessado em Janeiro de 2010 em http://www.vmware.com/products/thinapp/* .

VMWare, I. (2010). http://www.vmware.com.

VMWare, Inc. (2000). VMware technical white paper. Technical report.

XenSource, 2007. (s.d.). *XenEnterprise vs. ESX Benchmark Results: Performance Comparison of Commercial Hypervisors. 2007*. Acesso em 2007, disponível em http://blogs.xensource.com/simon/wpcontent/uploads/2007/03/hypervisor_performance_comparison_1_0_3_no_esxdata.pdf

Yu, Y., Guo, F., Nanda, S., Lam, L., & Chiueh, T. (2006). A Feather-weight Virtual Machine forWindows Applications. *Proceedings of the Second ACM/USENIX Conference on Virtual Execution Environments (VEE'06), June, 2006* .

Zhang, Y., Wang, X., & Hong, L. (2009). Portable Desktop Applications Based on P2PTransportation and Virtualization. *Proceedings of the 22nd Large Installation System Administration Conference (LISA'08) November 2008 . San Diego, CA: USENIX Association*, p. 133–144.

Autorizo a reprodução xerográfica para fins de pesquisa.

Bauru, _____/_____/____

_____
Assinatura

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo