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

Daniel Igarashi da Cruz

FLEXLAB: *Middleware* de virtualização de *hardware* para gerenciamento centralizado de computadores em rede

UNESP
2008

Daniel Igarashi da Cruz

FLEXLAB: *Middleware* de virtualização de *hardware* para gerenciamento centralizado de computadores em rede

Orientador: Prof. Dr. Marcos Antônio Cavenaghi

Dissertação de Mestrado elaborada junto ao Programa de Pós-Graduação em Ciência da Computação – Área de Concentração em Sistemas de Computação, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

UNESP

2008

"One machine can do the work of fifty ordinary men.
No machine can do the work of one extraordinary man."
Elbert Hubbard (1856 - 1915)

AGRADECIMENTOS

Agradeço ao meu orientador Prof. Dr. Marcos Antônio Cavenaghi pelo incentivo, inspirações, correções e adequações que permeiam todo este trabalho e que muito contribuíram para que este projeto fosse viabilizado mesmo diante de desafios, alguns além dos nossos recursos técnicos e financeiros, superados graças a sua interseção.

Aos meus queridos pais, José Pedro e Teresa Ayako, e minha irmã Juliana, por todo o esforço, dedicação e carinho que tiveram por mim durante toda a minha vida e por serem a minha maior referência de valores e ideais.

Ao meu amigo César de Souza Aguiar, pelo companheirismo nas madrugadas e finais de semana consumidos no desenvolvimento do projeto, por compartilhar idéias e por oferecer seu apoio às pesquisas.

À UNESP e ao Departamento de Computação da Faculdade de Ciências de Bauru, por oferecer a infra-estrutura para a realização dos experimentos necessários para o desenvolvimento do projeto.

À MStech, empresa que sempre ofereceu suporte, recursos e liberdade para que o projeto pudesse ser desenvolvido.

Ao Professor Eduardo Morgado, coordenador do Laboratório de Tecnologia da Informação Aplicada, grande amigo e incentivador.

Aos funcionários e professores da UNESP, pelo seu papel e sua representatividade no desenvolvimento da educação e da sociedade.

E por fim, agradeço a minha querida namorada e companheira Barbara De Franco, por sua motivação e pela ajuda oferecida nos experimentos realizados no projeto e, o mais importante, por ter sido compreensiva nos vários momentos em que estive ausente.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AMD-V: *Advanced Micro Devices – Virtualization Technology*

API: *Application Programming Interface*

ARM: *Advanced RISC Machine*

CAD: *Computer Aided Design*

CPL: *Current Privilege Level*

CPU: *Control Process Unit*

DDR: *Double Data Rating*

DHCP: *Dynamic Host Configuration Protocol*

DMA: *Direct Memory Access*

EFI: *Extensible Firmware Interface*

GPL: *GNU General Public License*

HLL: *High Level Language*

HT: *Hyper-Threading*

HVM: *Hybrid Virtual Machine*

IBM: *International Business Machine*

IDE: *Integrated Device Electronics*

IF: *Interruption Flag*

IP: *Internet Protocol*

ISU: *Imagem de Sistema Única*

JIT: *Just-in-time*

LAN: *Local Area Network*

LGPL: *Lesser GNU General Public License*

MAC: *Medium Access Control*

MB: *Mega Byte*

MIPS: *Microprocessor without Interlocked Pipeline Stages*

MMU: *Memory Management Unit*

MV: *Máquina Virtual*

NFS: *Network File System*

NOP: *No-Operation*

P2P: *Peert-to-Peer*

PNFS: *Parallel Network File System*

PSW: *Program Status Word*

PTE: *Page Table Entry*

PVFS: *Parallel Virtual File System*

PXE: *Pre eXecutable Environment*

RAM: *Random Access Memory*

ROM: *Read Only Memory*

SAD: *Sistema de Arquivos Distribuídos*

SMP: *Symmetric Multi-Processor*

SO: *Sistemas Operacional*

SOC: *Sistema Operacional Convidado*

SOH: *Sistema Operacional Hospedeiro*

SPARC: *Scalable Processor ARChitecture*

SSI: *Single System Image*

SVM: *Secure Virtual Machine*

TB: *Tradução Binária*

TC: *Translation Cache*

TFTP: *Trivial File Transfer Protocol*

ULA: *Unidade Lógica Aritmética*

USB: *Universal Serial Bus*

UUID: *Unique Universal Identifier*

VCPU: *Virtual CPU*

VGA: *Video Graphics Adapter*

VMCB: *Virtual Machine Control Block*

VMM: *Virtual Machine Monitor*

VT-i: *Virtualization Technology, Itanium Architecture*

VT-x: *Virtualization Technology, x86 Architecture*

WAN: *Wide Area Network*

LISTA DE FIGURAS

LISTA DE TABELAS

Resumo

O gerenciamento de um conglomerado de computadores em rede é uma atividade potencialmente complexa devido à natureza heterogênea destes equipamentos. Estas redes podem apresentar computadores com diferentes configurações em sua camada de *software* básico e aplicativos em função das diferenças de configuração de *hardware* em cada nó da rede. Neste cenário, cada computador torna-se uma entidade gerenciada individualmente, exigindo uma atividade manual de configuração da imagem de sistema ou com automatização limitada à camada de aplicativos. Tecnologias que oferecem gestão centralizada, como arquiteturas *thin-client* ou terminal de serviços, penalizam o desempenho das estações e oferecem capacidade reduzida para atender um número crescente de usuários uma vez que todo o processamento dos aplicativos dos clientes é executado em um único nó da rede. Outras arquiteturas para gerenciamento centralizado que atuam em camada de *software* são ineficazes em oferecer uma administração baseada em uma imagem única de configuração dado o forte acoplamento entre as camadas de *software* e *hardware*. Compreendendo as deficiências dos modelos tradicionais de gerenciamento centralizado de computadores, o objetivo deste trabalho é o desenvolvimento do FlexLab, mecanismo de gerenciamento de computadores através de Imagem de Sistema Única baseado em um *middleware* de virtualização distribuída. Por meio do *middleware* de virtualização do FlexLab, os computadores em rede de um ambiente são capazes de realizar o processo de *boot* remoto a partir de uma Imagem de Sistema Única desenvolvida sobre um *hardware* virtualizado. Esta imagem é hospedada e acessada a partir de um servidor central da rede, padronizando assim as configurações de *software* básico e aplicativos mesmo em um cenário de computadores com configuração heterogênea de *hardware*, simplificando o gerenciamento de computadores a uma única entidade gerenciável. Os experimentos do FlexLab mostraram que o tempo de *boot* de um computador utilizando a arquitetura virtualizada através de tradução binária dinâmica associada a um sistema de arquivos multicamada (camada de middleware e camada de máquina virtual) aumenta o tempo de iniciação em 37,5% em relação a um computador com *boot* normal (através do disco rígido local), com degradação de 5% na execução dos aplicativos sobre o *middleware* virtualizado. Com a associação do *middleware* de virtualização distribuída a um sistema de arquivos distribuídos baseado em pNFS/PVFS, o tempo de *boot* do FlexLab em um ambiente de 30 computadores é aprimorado em 37% e a degradação de desempenho de execução da Imagem de Sistema Única em função do *middleware* de virtualização se mantém entre 5% para um computador e 11% para 30 computadores. Os testes desenvolvidos pelo projeto mostram que os recursos de gerenciamento centralizado do FlexLab não penalizam a capacidade de escala do sistema uma vez que o *middleware* de virtualização atua de forma distribuída, portanto, desonerando o servidor da rede do processamento dos aplicativos dos computadores gerenciados.

PALAVRAS-CHAVE: Virtualização Distribuída, *Middleware* de Virtualização, Gerenciamento de Computadores, Máquinas Virtuais, Imagem de Sistema Única.

Abstract

Computer network management is a potentially complex task due to the heterogeneous nature of the hardware configuration of these machines. These networks may offer computers with different configuration in their basic software layer due to the configuration differences in their hardware layer and thus, in this scenario, each computer becomes an individual managed entity in the computer network and then requiring an individual and manually operated configuration procedure or automated maintenance restricted to application layer. Thin-client or terminal services do offer architectures for centralized management, however these architectures impose performance penalties for client execution and offer reduced scalability support in order to serve a growing number of users since all application processing is hosted and consume processing power of a single network node: the server. In the other hand, architectures for centralized management based on applications running over software layer are inefficient in offer computer management based on a single configuration image due to the tight coupling between software and hardware layers. Understanding the drawbacks of the theses centralized computer management solutions, the aim of this project is to develop the FlexLab, centralized computer management architecture through a Single System Image based on a distributed virtualization middleware. Through FlexLab virtualization middleware, the computers of a network environment are able to remote boot from a Single System Image targeting the virtual machine hardware. This Single System Image is hosted at a central network server and thus, standardizing basic software and applications configurations for networks with heterogeneous computer hardware configuration which simplifies computer management since all computers may be managed through a Single System Image. The experiments have shown that the boot over the virtualized architecture through dynamic binary translation associated with the multitier file system (middleware layer and software layer) is 37,5% slower than the regular local *boot* (through the local hard disk), and the applications run 5% slower over the virtualized middleware. With the combination of a virtualization middleware with a Distributed File System based on pNFS/PVFS, the *boot* time for a computer network environment with 30 computers is enhanced in 37% and performance degradation for the Single System Image due to the virtualization middleware é 5% for one computer and 11% for 30 computers. The tests developed by FlexLab Project show that the centralized management architecture does not sacrifice the system scalability since the virtualization middleware compromises the processing resources of computers in a distributed approach and the performance degradation varies according to the Distributed File System performance, which decreases as the number of computer increases.

KEYWORDS: Distributed Virtualization, Virtualization Middleware, Computer Management, Virtual Machines, Single System Image.

1.INTRODUÇÃO

1.1. Contextualização

A presença maciça de computadores nas organizações é um reflexo do seu impacto no aumento de produtividade e capacidade de gerar valor das empresas. Entretanto, o maior número de computadores nas organizações também criou demanda para ferramentas de gerenciamento de redes de computadores capazes de reduzir o custo total de propriedade (ELENBOGEN, 1999). A existência de redes locais de alta velocidade e processadores de baixo custo vêm favorecendo organizações a adotarem soluções de gerenciamento centralizado de computadores, como as soluções *thin-client* que exploram a arquitetura cliente-servidor (BARATTO, KIM & NIEH, 2005). Mesmo organizações como instituições de ensino, com laboratórios de informática com computadores em rede, podem se beneficiar amplamente de um planejamento e gerenciamento de computadores para promover um nível dinâmico de maturidade computacional (MORGADO, CRUZ & TWANI, 2008), onde as ferramentas de provisão e gerenciamento da configuração de *software* favorecem um ambiente de execução estável e padronizado.

Soluções de gerenciamento centralizado de computadores baseadas na arquitetura cliente-servidor, entretanto, oferecem limitações intrínsecas a sua arquitetura, com a maior parte do processamento dos clientes centralizado no servidor, o que limita a capacidade de se adicionar novos nós na rede e não favorecem o desempenho para a execução de aplicações que exijam cálculos matemáticos complexos, como aplicações para reprodução de vídeo. Outras arquiteturas para gerenciamento centralizado também esbarram no forte acoplamento entre o *hardware* e *software* de um computador, fazendo com que o gerenciamento dos computadores seja individualizado para cada conjunto de *hardware*/*software* e dificultando a sua execução em escala.

Embora diversas tecnologias de virtualização, como VMware (VMWARE, 1998), QEMU (BELLARD, 2004), XEN (BARHMAN, 2003) e VirtualPC (MICROSOFT, 2003), estejam disponíveis também para computadores pessoais, as

suas aplicações procuram explorar a utilização da virtualização em plataformas de grande porte, como servidores multi-processados, com o objetivo de maximizar o seu uso por meio da execução de múltiplas instâncias de sistemas operacionais sobre sua plataforma. Neste contexto, a virtualização é usada não como ferramenta de gerenciamento, mas sim como ferramenta de otimização para *data-centers* e outros ambientes de computação de alto-desempenho. Por outro lado, as redes locais de alta velocidade, a redução no custo dos processadores e a possibilidade de aplicação de técnicas de computação em *grid* e *cluster* em computadores *commodity* criam condições para que as tecnologias de virtualização sejam aplicadas também dentro do contexto de gerenciamento centralizado de computadores (KRSUL, et al. 2004). Neste cenário, a virtualização pode ser aplicada em computadores pessoais como forma de se abstrair as complexidades de configuração de uma rede adotando-se um único conjunto de *hardware* padronizado, o da máquina virtual. Com isso, todos os computadores utilizam uma única imagem padronizada de *software*, desenvolvida a partir do conjunto do *hardware* da máquina virtual (MV).

As atuais soluções de mercado que empregam o conceito de virtualização para gerenciamento de computadores utilizam a virtualização para a criação de uma infra-estrutura virtual de *desktops* (VMWARE, 2007), em uma arquitetura similar a arquitetura *thin-client* e que, portanto, oferece as mesmas limitações de escala em função da maior parte do processamento ficar localizado no servidor central de virtualização. Neste caso a virtualização é utilizada para se criar *desktops* virtuais no servidor de virtualização e estes *desktops* são utilizados por meio de um cliente *Web* pelos computadores da rede. Como estes computadores agem apenas como terminais, eles podem compartilhar a mesma infra-estrutura com *thin-clients.*

Outras soluções, como o Citrix Provisioning Server for *Desktop*s (CITRIX, 2008) e o Ardence *Desktop* Streaming (ARDENCE, 2007) propõem a virtualização do disco rígido das estações e mecanismo de *boot* remoto. Neste caso, apenas o disco rígido é virtualizado por meio de um arquivo binário único que é montado pela rede como um dispositivo de acesso a blocos pelas estações em tempo de *boot*, como se fosse um disco rígido normal. Por meio do protocolo PXE (INTEL, 1999) os computadores realizam o *boot* utilizando remotamente este arquivo que representa o disco rígido, como se este estivesse no computador. Esta solução oferece um ótimo compromisso entre desempenho e requisitos de *hardware* oferecendo uma grande capacidade de escala, porém, também sofre com o forte acoplamento entre *hardware* e

software, o que pode exigir que cada computador de uma rede heterogênea de *hardware* tenha a sua própria Imagem de Sistema Única (ISU).

A proposta do FlexLab é a combinação das técnicas de virtualização com a utilização de sistemas de arquivos distribuídos, utilizados em *clusters* de computadores, e que permite a criação de uma solução de gerenciamento centralizada com processamento distribuído nos computadores da rede sem a necessidade de que cada computador tenha sua própria imagem individualizada de *software*. Esta abordagem representa uma nova perspectiva em soluções para gerenciamento de computadores e propõem uma nova utilização para a tecnologia de virtualização.

1.2. Motivação

Tendências recentes da indústria, como a consolidação de servidores e ambientes mais tolerantes a falhas, associados à diminuição do custo de processadores e memórias, o surgimento dos processadores de múltiplos núcleos para computadores pessoais e a necessidade de aumentar a eficiência operacional de *data-centers* têm motivado o ressurgimento do interesse nas tecnologias de virtualização de *hardware*, particularmente as máquinas virtuais (ROSENBLUM, 2004). Dentre as diversas possibilidades da virtualização de *hardware*, o melhor aproveitamento do poder computacional ocioso dos equipamentos é apenas uma das vantagens desta tecnologia (SMITH & NAIR, 2005).

A virtualização de *hardware* permite, resumidamente, que diversas instâncias de sistemas operacionais distintos ou iguais operem concorrentemente sobre um mesmo conjunto de *hardware*, sendo que cada sistema operacional "enxerga" o computador virtualizado como sendo real e de uso exclusivo da sua instância de execução. Para as aplicações, todo o *hardware* virtualizado apresenta as mesmas interfaces e funcionamento do real, o que permite a utilização desta tecnologia em diversas soluções sem a necessidade de alteração do código fonte da aplicação.

Além de permitir a execução de múltiplas instâncias de máquinas virtuais sobre um mesmo equipamento, a tecnologia de virtualização também cria uma interessante padronização de *hardware*, abstraindo as suas complexidades do sistema operacional e da camada de aplicações (GOLDBERG, 1973). Esta característica peculiar às máquinas virtuais, associada aos sistemas de arquivos distribuídos, permite a proposta de novas soluções para gerenciamento de computadores envolvendo virtualização onde é

desejável a padronização de *hardware*. Nestes cenários, é possível a utilização de uma imagem de *software* única, onde uma configuração de *software* é capaz de atender a todos os computadores de um determinado ambiente (ZHOU, et al. 2006), simplificando o gerenciamento e a disponibilização destes computadores.

1.3. Objetivos

O objetivo deste trabalho é desenvolver o FlexLab, proposta de *middleware* de virtualização distribuída para gerenciamento de computadores, e avaliar o seu desempenho e a aplicabilidade da virtualização em um ambiente distribuído, onde ao invés de concentrar diversas máquinas virtuais em um único nó, os nós de virtualização são distribuídos entre estações de pequeno porte, neste caso, computadores pessoais com processadores de arquitetura x86. Embora o foco deste trabalho seja avaliar a utilização da virtualização distribuída como *middleware* para gerenciamento de computadores, o FlexLab também serviu de base para outro projeto de pesquisa concluído que estudou e avaliou a aplicação do *middleware* e o impacto dos diferentes mecanismos de Sistemas de Arquivos Distribuídos como ferramenta para a criação de *clusters* multiuso (AGUIAR, 2008).

Este projeto de pesquisa tem por objetivos específicos:

- Desenvolver um *middleware* que funcione como uma camada de virtualização entre o *hardware* real e a Imagem de Sistema Única contendo o sistema operacional e os aplicativos *desktop*;
- Desenvolver para o *middleware* a capacidade de execução de uma máquina virtual sobre um sistema de arquivos distribuído, permitindo assim que a arquitetura da aplicação possa explorar a execução distribuída de máquinas virtuais (MVs);
- Desenvolver um mecanismo de *boot* remoto para o *middleware*, permitindo assim que o computador cliente gerenciável não precise armazenar nem o *middleware* e nem a Imagem de Sistema Única;
- Realizar experimentos para analisar o desempenho da arquitetura proposta pelo *middleware* de virtualização distribuída;
- Realizar experimentos para analisar o desempenho da camada de virtualização em função do tipo de mecanismo de virtualização adotado.

1.4. Estrutura da Monografia

A monografia está organizada em sete capítulos e os assuntos abordados neste trabalho de pesquisa estão divididos da seguinte forma:

Capítulo 1: Contém a introdução ao trabalho e apresenta a contextualização da pesquisa, sua motivação e os objetivos pretendidos.

Capítulo 2: Descreve as principais técnicas de virtualização em nível de *hardware* e as arquiteturas de máquinas virtuais, abordando as principais dificuldades dos mecanismos de virtualização sobre a arquitetura x86, a mais comumente utilizada em computadores pessoais e que é o objetivo desta pesquisa.

Capítulo 3: Apresenta uma comparação entre as técnicas de virtualização e as máquinas virtuais clássicas estudadas neste trabalho.

Capítulo 4: Apresenta trabalhos relacionados e conceitos e mecanismos utilizados para o gerenciamento de máquinas virtuais distribuídas em ambientes de *Cluster*, apresentando brevemente os modelos de Sistemas de Arquivos Distribuídos e formas para se aprimorar o desempenho de acesso concorrente a blocos de dados.

Capítulo 5: Descreve o FlexLab, *middleware* de virtualização distribuída para gerenciamento de computadores, sua aplicação e detalha o seu funcionamento.

Capítulo 6: Apresenta os resultados dos experimentos e realiza uma análise dos resultados obtidos.

Capítulo 7: Conclui a dissertação apresentando as contribuições deste trabalho e apresenta os trabalhos futuros com base nos estudos desenvolvidos a partir do FlexLab.

2.ARQUITETURA DE MÁQUINAS VIRTUAIS

2.1. Considerações Iniciais

Os sistemas de máquinas virtuais (MV) representaram um grande avanço no desenvolvimento de novos sistemas computacionais, oferecendo uma eficiente transcrição de sistemas computacionais completos e assim, permitindo o desenvolvimento de sistemas multi-programados e multi-processados (GOLDBERG, 1973). De certa forma, todos os benefícios até então desfrutados apenas por desenvolvedores de aplicação, como a multi-programação (SILBERSCHATZ, et al. 1991), através dos sistemas operacionais modernos, passaram a ser oferecidos também para os desenvolvedores de sistemas com o multi-processamento (POPEK, 1975) criado pela virtualização. A máquina virtual introduz também uma capacidade interessante de oferecer abstração de *hardware*, tanto através de máquinas virtuais em nível de *software* como Java e Inferno (SIRER, et al. 1999), como com a utilização de mecanismos de virtualização em nível de *hardware* através de monitores de máquina virtual como VMware, VirtualPC e KQEMU (BELLARD, 2005). Este capítulo apresenta uma breve revisão sobre os mecanismos de virtualização, descrevendo os modelos utilizados em arquitetura virtualizáveis e não virtualizáveis, como a arquitetura x86, as técnicas de virtualização e sua aplicação ao desenvolvimento do *middleware* do FlexLab.

2.2. Abstração de *Hardware* por meio de Máquinas Virtuais

O uso das máquinas virtuais, de forma geral, introduz a capacidade de:

- Aprimorar o desenvolvimento e teste de *software*;
- Operar diversos sistemas operacionais simultaneamente;
- Permitir o funcionamento de MVs com diversas configurações de E/S, memória, etc;

- Aferição de sistemas operacionais;
- Utilização de discos virtuais para armazenamento persistente de dados;
- Isolação de aplicações através de endereçamento de memória;
- Execução segura de aplicações com isolação entre processos;
- Acesso a recursos de E/S do *hardware* real.

Para promover a virtualização de *hardware*, ao fim da década de 60 o mecanismo de *Virtual Machine Monitors* (VMM) foi introduzido como forma de oferecer uma pequena camada de abstração de *hardware*, permitindo que diversos sistemas operacionais pudessem funcionar concorrentemente sobre um mesmo *hardware* real. Cada uma dessas MVs oferecia uma interface virtualizada de execução suficientemente similar ao *hardware* real sob o qual funcionava, o que garantia um nível suficiente de similaridade com o sistema real para uma execução consistente das aplicações sem a necessidade de modificá-las em função da virtualização.

O mecanismo de VMM é bastante flexível e permite a virtualização de diversas arquiteturas de computadores, desde computadores com arquitetura x86 de baixo custo até arquiteturas de alto desempenho como os processadores vetoriais (BELLARD, 2000).

Porém, com o surgimento dos sistemas operacionais modernos e com suporte a execução de multi-tarefas, associado ao notável crescimento do poder computacional dos computadores comerciais, a importância das MVs foi decaindo, passando de uma área promissora para uma mera curiosidade acadêmica por décadas. Muito do foco na última década foi dedicado as MVs em nível de *software*, como mostra a Figura 1, empregando os mesmos conceitos de virtualização, porém para um escopo de execução limitado a camada de aplicativos, e não de sistema operacional.

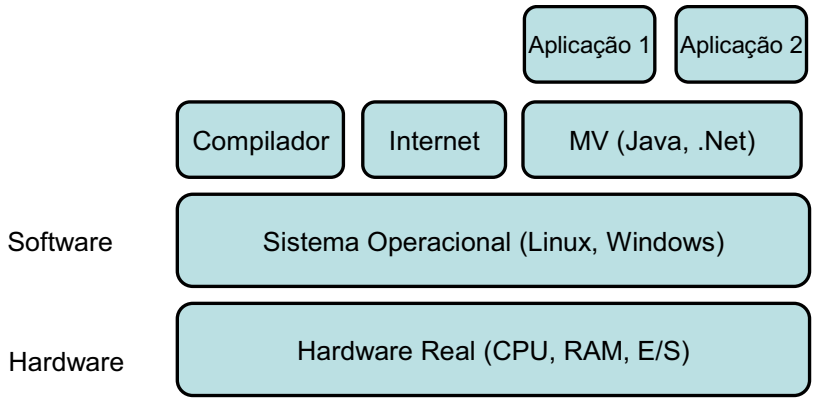

Figura 1: A estrutura de um sistema computacional moderno com a utilização de uma máquina virtual em nível de *software* (ROSEMBLUM, 2004)

Passada a década de 90, tanto o mundo acadêmico quanto as grandes empresas do setor passaram a manifestar novamente interesse pelas máquinas virtuais em nível de *hardware*. Para efeito de simplificação, a menos que seja indicado ao contrário, o trabalho sempre vai se referir às máquinas virtuais em nível de *hardware* como MVs.

O ressurgimento das MVs se deu na área acadêmica através dos sistemas de processamento massivamente paralelo (MPP), difíceis de programar e que necessitavam de um sistema operacional diferente dos sistemas comerciais. Com as MVs, os pesquisadores puderam utilizar estes sistemas através de aplicações desenvolvidas para os sistemas operacionais convencionais, que por sua vez executavam sobre uma MV representando os sistemas convencionais com execução direta sobre os computadores de arquitetura MPP (FARREL, 1995).

A combinação de sistemas operacionais modernos com o *hardware* de baixo custo e grande capacidade computacional fez com que a proliferação de máquinas virtuais fosse possível. Com o desperdício computacional cada vez maior e com as diversas fragilidades impostas pelos sistemas operacionais modernos, a abordagem de virtualização parece ser uma proposta eficiente para se resolver diversos problemas em troca de uma deterioração no desempenho da execução virtualizada (MENON at el. 2005) em função da tecnologia e abordagem adotada. Desta forma, como mostra a Figura 2, a virtualização é um importante componente para promover a abstração do *hardware*, mantendo a compatibilidade com a ISA de uma determinada arquitetura e permitindo a execução de múltiplas instâncias de sistemas operacionais.

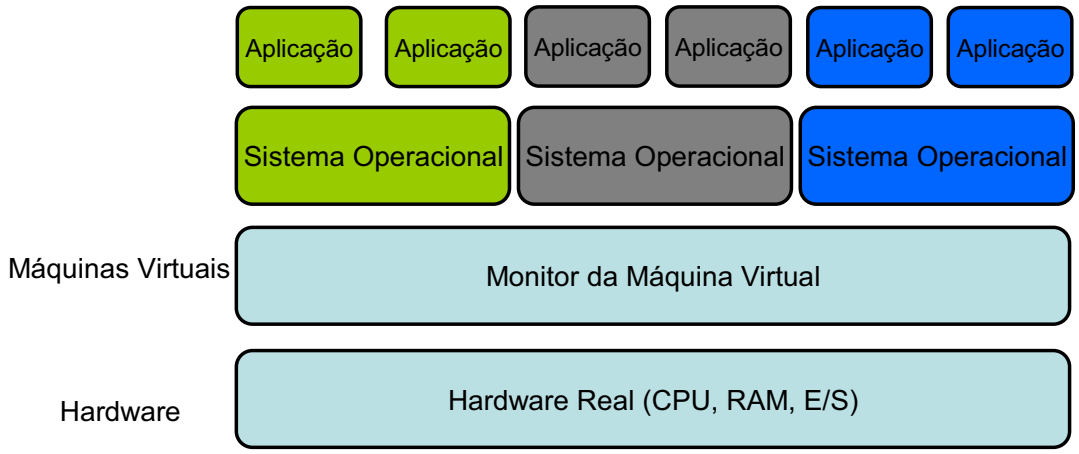

Figura 2: Monitor da Máquina Virtual, mecanismo sobre as quais diversas máquinas virtuais podem executar sobre um mesmo *hardware* real (ROSEMBLUM, 2004).

Embora sejam sistemas complexos, os sistemas computacionais são baseados em interfaces de comunicação bem definidas, criando níveis coerentes de abstração (SMITH & NAIR, 2005) que permitem a redução da complexidade durante a programação. Por exemplo, embora os circuitos de controle de uma interface de E/S sejam complexos, a definição clara de seu fluxo de operação e mecanismos de funcionamento, endereçamento e bits de controle permite que a sua programação seja simples e feita através de chamadas a interfaces padrão de programação, reconhecidas pela arquitetura do processador e também pelo sistema operacional. Para o programador, não importa a complexidade da implementação eletrônica, a forma de programação é sempre baseada em uma interface bem definida de controle.

Um conjunto de instruções de uma arquitetura (ISA) também segue estes mesmo princípios de abstração. Ao longo do tempo, diversas empresas vêm desenvolvendo *hardware* e *software* sob a arquitetura IA-32, que é a arquitetura x86 originalmente proposta pela Intel para os microprocessadores. Empresas, como Microsoft e AMD, desenvolvem produtos de acordo com a ISA IA-32, e mesmo sendo produtos diferentes, todos observam as disposições da IA-32 e em função disto todos os produtos interoperam. Entretanto, o nível de definição das ISA é limitado a um conjunto de instruções específicas da arquitetura de um processador, ficando os demais subsistemas e periféricos de E/S de fora de uma padronização mais formal quanto à interface de seus componentes. Esta falta de total definição de parâmetros, e até mesmo a abrangência de sua complexidade, ainda não permitem total interoperabilidade entre os diversos fornecedores. O escopo deste estudo é válido para a arquitetura x86, embora a

virtualização seja uma técnica também amplamente empregada em outras ISAs (FARELL, HIERONSKA & KORDA, 1995).

A abstração de *hardware* deve ser suficientemente abrangente com relação a sua ISA. Esta é uma forma de garantir que o conjunto de instruções de uma determinada arquitetura esteja integralmente transcrita dentro do conjunto de instruções expressadas por uma MV. Dessa forma, tanto aplicações quanto sistemas operacionais compilados para esta ISA funcionarão corretamente e de forma previsível dentro da MV. Uma discussão de uma MV é necessariamente uma discussão sobre arquitetura de computadores. Para ser confiável, uma MV precisa exprimir a especificação formal das interfaces do sistema, incluindo o comportamento lógico de todo os subsistemas. A Figura 3 mostra os níveis de abstração e as interfaces implementadas para exprimir o funcionamento lógico da arquitetura x86, também aplicável a diversas arquiteturas.

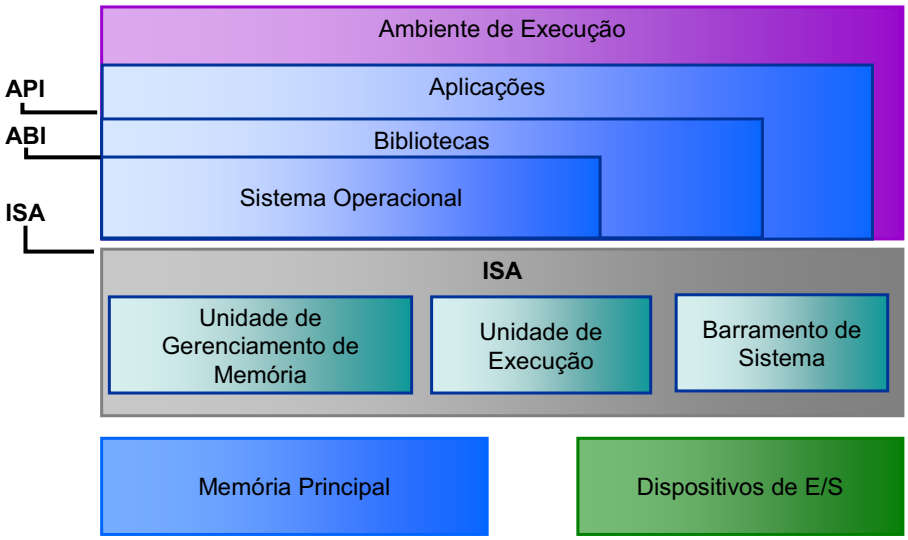

Figura 3: Arquitetura de um sistema com a divisão *Hardware*/*Software* marcada pela ISA (SMITH & NAIR, 2005).

Desta forma, cada subsistema pode ser adicionado ao sistema principal de execução de forma a ser reconhecido pelo SO e pelas aplicações como parte deste sistema, permitindo também a comunicação de dados através de interfaces que definem o funcionamento lógico de cada sistema. Para uma MV, a fidelidade da transcrição das interfaces de cada sistema e subsistema é fundamental para oferecer uma correta abstração de *hardware*. Para a construção de uma MV, três interfaces são de fundamental importância (BREY, 2003):

- O conjunto de instruções da arquitetura, ou *instruction set architecture* (ISA);
- A interface binária das aplicações ou *application binary interface* (ABI);
- A interface de programação da aplicação ou *application programming interface* (API).

Essas interfaces serão detalhadas nas seções a seguir.

2.3. Abrangência de Transcrição da ISA

A ISA marca a divisão entre o *hardware* e o *software*. A ISA é composta na maioria das arquiteturas de dois conjuntos; um conjunto de instruções para execução em modo usuário, ou ISA Usuário, e que oferece um subconjunto de instruções visível às aplicações de usuário, e também de um conjunto de instruções ISA Supervisor, que constituem o conjunto completo de instruções visível ao Sistema Operacional e aos mecanismos de controle e gerenciamento dos recursos de *hardware* apenas.

Processadores modernos oferecem também um conjunto de instruções especialmente desenvolvidas para aprimorar o funcionamento da virtualização, reduzindo a complexidade do *software* da VMM (NEIGER, et al. 2006). Um exemplo é a tecnologia Intel Virtualization, disponível nas versões VT-x para processadores x86 e VT-i para processadores Itanium de 64-bits (ROSENBLUM, 2004). Estas tecnologias têm como objetivo superar as dificuldades de virtualização presente em cada uma destas arquiteturas de forma a oferecer um conjunto adicional de instruções que simplificam as operações realizadas pela VMM.

2.4. Interface Binária de Aplicação

A ABI oferece uma interface para que aplicações tenham acesso aos recursos de *hardware* através da ISA Usuário e a interface de chamada de sistema. A ABI não oferece as instruções de sistema e todos os acessos aos recursos de *hardware* são feitos através de chamadas aos procedimentos de acesso a *hardware* do SO, como mostra a Figura 3. Esta interface do SO é suficientemente abstraída para cada conjunto de subsistemas (placa de rede, interface com portas seriais, USB, etc) de forma a permitir

que cada aplicação consiga acessar os subsistemas de *hardware* sem ter conhecimento completo sobre a sua implementação.

2.5. Máquinas Virtuais de Linguagens de Alto Nível

Máquinas Virtuais de Linguagens de Alto Nível (High Level Language – HLL), ou máquinas virtuais em nível de linguagem, como o ambiente de execução Java (SUN MICROSYSTEMS, 1999) e o .Net *Framework* (MICROSOFT CORPORATION, 2002) emulam uma arquitetura padrão sobre diversas outras arquiteturas reais de *hardware*, desde que haja um porte específico do ambiente de execução virtual para estas diversas arquiteturas. Esta abordagem oferece uma série de vantagens do ponto de vista de interoperabilidade de *software* entre diversas arquiteturas distintas.

Embora este tipo de MV não seja de interesse principal para este estudo, elas também apresentam características comuns aos demais tipos de MVs, como a necessidade de implementarem uma camada de abstração capaz de expor uma ISA.

A definição de uma arquitetura virtual completa para uma MV de Linguagem de Alto Nível não corresponde diretamente a nenhuma arquitetura real de *hardware*, ao contrário, procura definir um conjunto de interfaces capaz de exprimir todos os recursos disponibilizados pelo ambiente de execução virtual para um processo. Na comparação da Figura 4, o lado direito da figura mostra mais claramente o processo. Em uma aplicação com execução nativa (lado esquerdo da Figura 4), o código-fonte compilado dá origem a um código objeto ligeiramente similar ao da máquina, porém mais abstraído em linguagem de programação, e então este código objeto dá origem aos binários compatíveis com as instruções da arquitetura da máquina.

No compilador para uma MV HLL (lado direito da Figura 4), o código fonte gerado para uma ISA virtual, que especifica as interfaces de programação da MV, é compilado e este código intermediário a linguagem de montagem da MV é distribuído para execução em diversas plataformas.

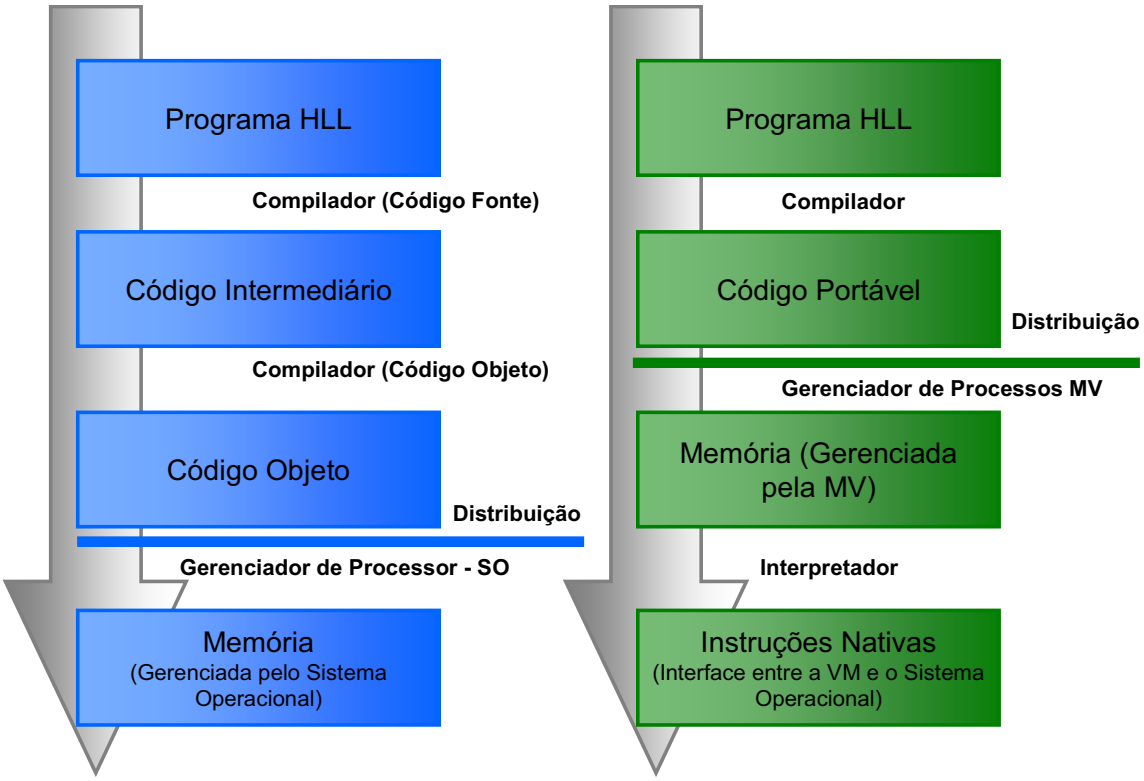

Figura 4: Comparação entre a execução gerenciada pelo Sistema Operacional e um ambiente de MV HLL com um gerenciador próprio.

A MV HLL executa sobre o sistema operacional como um processo, e dentro deste processo todo o ambiente virtual é emulado, de forma a garantir as aplicações dentro deste ambiente total isolamento com relação ao sistema operacional do computador. Esta abordagem, entretanto, não deixa de exigir para a MV HLL uma definição rigorosa da ISA emulada, garantindo a mesma especificação para ISA para execução sobre diferentes plataformas.

2.6. Máquinas Virtuais de Sistema

Uma máquina virtual de sistema implementa um ambiente completo na qual o sistema operacional e os processos podem co-existir. Através de máquinas virtuais, um único *hardware* pode suportar diversos sistemas completos operando concorrentemente, sendo que cada sistema, a rigor, não tem conhecimento da execução do outro sistema.

As MVs de sistema surgiram durante a década de 1960 e começo da década de 1970. Naquele tempo, o contexto da computação era dominado pelos *mainframes*, servidores de grande porte, caros e geralmente com uso compartilhado entre diversos usuários. A conveniência da tecnologia de MVs permitia que múltiplos usuários

utilizassem os *mainframes* com diferentes SOs. À medida que os *hardware*s ficaram mais acessíveis e mais poderosos, a tecnologia de virtualização foi lenta e consistentemente abrangendo também os computadores comerciais de baixo custo equipados com processadores x86.

A possibilidade de virtualização da arquitetura x86 apresenta vantagens interessantes, pois se trata de uma arquitetura com grande adoção de mercado, de baixo custo e com diversos *software*s já desenvolvidos para ela. É também a arquitetura mais presente nas diversas áreas da economia, como ciências, educação, indústria em geral e mercado financeiro (ZHAO, et al. 2006).

Provavelmente a aplicação mais importante das MVs de Sistema seja a isolação de aplicações, permitindo que múltiplos sistemas funcionem concorrentemente sobre o mesmo *hardware*. Em um sistema virtualizado através de VMM, as interfaces virtuais de cada instância da MV são replicadas para cada instância, permitindo que cada SO enxergue o computador como sendo inteiramente seu (LEVASSEUR, et al, 2005). Desta forma, cabe a VMM ter acesso, controlar e gerenciar todos os recursos de *hardware*. Um SO convidado (executando sobre uma instância da MV) possui autonomia para gerenciar suas aplicações, mas a sua execução dentro do escalonador de instruções do processador é controlada pela VMM. Quando o SO convidado executa uma instrução privilegiada com interatividade direta sobre um recurso possivelmente compartilhado de *hardware*, a VMM intercepta a operação e avalia as condições de execução segura. Se a execução for segura, a VMM executa a instrução protegida como se fosse o SO convidado, retornando o resultado novamente para o mecanismo de *write back* do processador virtualizado. Para as aplicações executando dentro do SO convidado, toda a operação é transparente.

2.7. Máquinas Virtuais de Sistema Clássicas

Atualmente, diversos mecanismos de virtualização aplicam técnicas que oferecem os recursos de abstração de *hardware* e transcrição de instruções de uma ISA, sendo, portanto, candidatos a aplicação no *middleware* do FlexLab. Entretanto, em função do mecanismo de virtualização adotado, características como desempenho, fidelidade e segurança, todas necessárias para o funcionamento esperado do *middleware*

de virtualização do FlexLab, podem não estar presentes em função dos desafios apresentados pela arquitetura x86 de processadores aos mecanismos de virtualização. A virtualização através de VMMs, também consideradas Máquinas Virtuais de Sistema Clássicas, envolve um conjunto de definições e características necessárias para o funcionamento do FlexLab e cujo entendimento é necessário para a definição do modelo de virtualização a ser utilizado pela proposta de *middleware* de virtualização, conforme o trabalho apresenta nesta seção.

Do ponto de vista das aplicações em execução dentro do SO Convidado, todos os mecanismos de virtualização apresentam o mesmo comportamento (ou seja, o comportamento de um computador real), e os detalhes de implantação variam de acordo com a técnica adotada de virtualização. Os modelos clássicos de virtualização (GOLDBERG, 1973) previam a VMM funcionando diretamente sobre o *hardware*, com cada MV funcionando diretamente sobre a VMM. Toda a VMM executa com o maior nível de privilégio no sistema, enquanto que as demais camadas de *software* (MV, SO Convidado, Aplicações) possuem gradualmente menores níveis de privilégio.

De acordo com a definição de Popek e Goldberg (POPEK & GOLDBERG, 1974), três características são essenciais para um *software* ser considerado uma VMM, a saber:

1. **Fidelidade:** Com exceção da degradação de desempenho, o *software* executando sobre a VMM deve possuir o mesmo comportamento do *software* executando sobre o *hardware* real, mesmo em um ambiente onde haja concorrência entre múltiplas instâncias de MVs em execução – este requisito faz com que os Sistemas Operacionais de tempo real não sejam qualificados também como VMMs;
2. **Desempenho:** A grande maioria das instruções do SO convidado deve executar diretamente sobre o processador real sem a intervenção direta da VMM – esta característica faz com que os emuladores não sejam também qualificados erroneamente como uma VMM, embora sejam capazes de introduzir um nível de virtualização da arquitetura;
3. **Segurança:** A VMM deve ser a responsável por gerenciar o acesso e uso de todos os recursos de *hardware* – desta forma, nenhuma aplicação executando sobre uma MV pode acessar diretamente os recursos de *hardware* previamente alocados para uma outra instância de MV.

A MV é o ambiente criado pela VMM, como mostra a Figura 5, servindo também para a criação de um sistema com suporte a multi-tarefa, uma importante característica mantida pela VMM e que permite com que ela suporte sistemas operacionais modernos, como Linux e Microsoft Windows.

Embora o modelo de virtualização de Popek e Goldberg parta de afirmações simplificadas a respeito da arquitetura de computadores da terceira geração, este modelo de virtualização proposto é bastante conveniente para analisarmos a eficiência de uma arquitetura quanto à possibilidade de virtualizá-la.

Dentro deste modelo, ambos determinaram quais são as principais características que uma ISA deve apresentar para oferecer suporte a virtualização atendendo às necessidades inerentes de um sistema virtualizado. Além disso, a MV também pode apresentar um ou mais processadores, diversos dispositivos de E/S, segmentação e paginação de memória e suporte a memória virtual.

Uma MV é apresentada como sendo composto de um processador clássico, que disponibiliza dois modos de execução: modo supervisor e modo usuário. No modo supervisor, o escalonador de instruções pode executar todas as suas instruções disponibilizadas pela arquitetura. Em modo usuário, algumas instruções privilegiadas possuem sua execução negada. Esta característica de execução é comum a todos os computadores derivados da terceira geração, como os baseados na arquitetura x86.

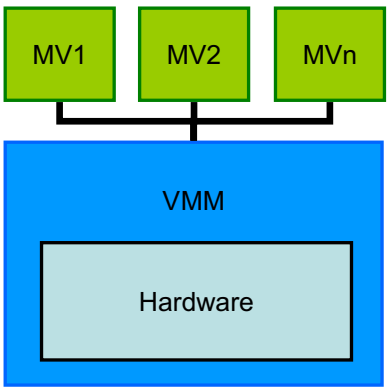

Figura 5: Máquina Virtual Clássica de acordo com a proposta de Goldberg (GOLDBERG, 1973).

O processo de virtualização parte do princípio de que existe um número finito de estados para a MV, sendo que cada estado possui quatro componentes que o qualificam: Instrução de memória *E*, modo do processador *M*, contador de programa *P* e registrador de realocação *R*, que indica qual a posição de memória equivalente ao "zero" para determinação do *off-set* para a área de memória virtual, configurando assim como um

registrador para realocação da área limite de memória da MV. Desta forma, o estado da MV é dado por:

$S = <E, M,P,R>$

O registrador E aponta para a *i-ésima* posição da área de dados da MV no formato *E[i]* somente se *E'[i]* para qualquer $0 \leq i < q$. Por exemplo, se for necessário que a MV acesse toda a área de memória do computador, então o registrador de realocação *R* receberá *a posição relativa de memória 0* (em relação a VMM) e o limite superior de memória será *q-1*. Desta forma, se uma instrução gera um endereço *a* qualquer, seu processamento é dado por:

if $a + 1 \geq q$ **then** ***memorytrap*** **else**
if $a > b$ **then** *memorytrap*
else *do E[a + 1];*

O contador de programa *P* é um endereço relativo para o conteúdo apontado por *R*, que age como um índice dentro do registrador *E* indicando qual a próxima instrução a ser executada dentro da MV. O estado de *S* indica o estado atual do sistema real do computador, e não da MV. O conteúdo desta tripla também é chamado de PSW, ou palavra de *status* do programa, desta forma, troca de contextos entre VMs são arbitradas pela VMM com operações *store old-PSW* e *fetch new-PSW* para cada estado *S[i]*.

Cada componente de *S* pode utilizar somente um número finito de valores, que podemos definir como um conjunto de estados *C* sem perda de generalização (GOLDBERG, 1973). Desta forma, uma instrução *i* é uma função que define um estado C_s inicial e um estado C_f final onde:

$i: C_s \rightarrow C_f,$

Por exemplo, $i(S_1) = S_2,$ ou $i(E_1, M_1, P_1, R_1) = (E_2, M_2, P_2, R_2)$.

Embora sejam naturalmente complexos, os sistemas computacionais em geral revelam um mecanismo de certa forma bastante simples de proteção ao redor do conceito de execução em modo usuário ou modo supervisor, além de um mecanismo de

endereçamento e alocação de memória simples, construído em torno do sistema de realocação de limites (representado no nosso sistema virtualizado pelo registrador *R*).

A VMM do tipo "*trap-and-emulate"* era o modelo mais comum de virtualização (ADAMS & AGESEN, 2006), principalmente no período de 1970, quando havia sido proposta originalmente por Goldberg para os computadores de terceira geração.

2.8. Comportamento das Instruções

A possibilidade de virtualização de uma arquitetura depende de uma função definida pelo seu estado *S*. Uma instrução é privilegiada se e somente se para qualquer par de estados S_1= <e, s, p, r> e S_2 <e, s, p, r> na qual $i(S_1)$ e $i(S_2)$ não causam um *trap* de memória e $i(S_2)$ causa um *trap* e $i(S_1)$ não, onde S_1 é uma instrução de modo supervisor e S_2 de modo usuário. O *trap* que ocorre dentro destas circunstâncias é chamado de *trap de uma instrução privilegiada.*

Assim, de acordo com a definição, uma instrução privilegiada precisa ser executada através do mecanismo de *trap*, o que elimina a possibilidade de utilização de outras técnicas, como simplesmente atribuir a esta instrução privilegiada uma equivalente a *NOP*. Uma instrução NOP que não causa *trap* pode ser chamada simplesmente de *NOP em modo usuário* (POPEK & GOLDBERG, 1974).

Alguns exemplos de instruções privilegiadas e computadores de terceira geração:

(1) **if** *M = s* **then** *load_PSW* // IBM System/360 LPSW
else *trap;*

(2) **if** *M = s* **then** *load_R* // Honeywell 6000 LBAR, DEC PDP-10
else *trap;*

É particularmente interessante também definir os tipos de *traps*. Um *trap* de memória é causado, como pode-se inferir logicamente, por uma instrução que requisita um endereço de memória que está além dos limites definidos no registrador de realocação do endereço de limite da memória, o registrador *R*.

if $a + l \geq q$ **then** *trap;*

if $a \geq b$ **then** *trap*

Outro tipo de *trap* é o *trap* de instruções privilegiadas, em função de uma instrução a ser executada em modo supervisor ou modo privilegiado.

As instruções privilegiadas dizem respeito a características específicas da máquina e não dizem respeito ao ambiente de execução virtual. Outro grupo importante de instruções são as instruções sensíveis, as quais podem ser classificadas de duas maneiras:

1. Instruções Sensíveis ao Controle: Instruções que podem mudar a quantidade de memória e recursos disponíveis, ou afetam o modo do processador sem causar uma seqüência de *trap* de memória;
2. Instruções sensíveis ao Comportamento: Instruções que podem mudar o estado do processador ou a área de endereçamento da VMM em função da resposta obtida por uma determinada operação. Um exemplo de instrução sensível ao contexto são as instruções que fazem referência a um endereço real de memória.

Uma instrução é sensível ao controle se existe um estado $S_1 = <e_1, m_1, p_1, r_1>$ e $i(S_1) = S_2 = <e_2, m_2, p_2, r_2>$ tal que $i(S_1)$ não sofre um *trap* de memória e nem (a) $r_1 \neq r_2$ ou (b) $m_1 \neq m_2$, ou ambos.

Em uma definição intuitiva de uma VMM, define-se que o controle completo sobre os recursos do sistema é um requisito. Entretanto, instruções sensíveis ao controle são instruções que podem potencialmente afetar este controle dos recursos.

Instruções sensíveis ao comportamento ocorrem normalmente quando o efeito da execução de uma instrução depende do valor apontado pelo registrador de realocação de limite de memória, ou seja, depende de um valor armazenado em uma região de memória real ou depende do modo de operação do processador. Exemplos de instruções sensíveis ao comportamento são instruções que têm como parâmetro um endereço de uma instrução anterior.

2.9. Virtualização por meio de VMM Clássica

Mesmo nas MVs mais recentes, a VMM é a porção de *software* que exerce o papel de *Programa de Controle.* Este programa exibe certas características e é composto por diversos módulos que permitem a virtualização da arquitetura x86, que podem ser representados por três grupos diferentes (GOLDBERG, 1973).

O primeiro grupo é um *Dispatcher D*. A sua instrução inicial *P* é alocada na posição 1. Este mecanismo despachante pode ser considerado um módulo de controle de alto nível e decide quais outros módulos devem ser chamados. O segundo grupo é um *Alocador*. São as tarefas executadas pelo Alocador que decidem quais recursos de *hardware* serão oferecidos para a MV. No caso para a execução de uma única VM, o Alocador precisa manter a VM e a VMM isoladas. No caso de um ambiente com múltiplas MVs executando concorrentemente, é o Alocador que gerencia a alocação de recursos, evitando que um recurso seja compartilhado simultaneamente por duas MVs. O Alocador é chamado pelo Módulo Despachante sempre que houver uma tentativa de execução de uma instrução privilegiada na qual seu resultado pode provocar mudanças nos recursos da máquina dentro de um ambiente de uma MV. Um exemplo é uma instrução que escreve no registrador *R*. Se o processador for tratado como um recurso (no caso de um sistema multiprocessado), uma instrução de *halt* seria outro exemplo.

O terceiro módulo no programa de controle pode ser chamado de *Interpretador*, já que é responsável por definir uma rotina de tratamento para cada instrução privilegiada que realiza um *trap*. Cada rotina do interpretador tem como objetivo simular o efeito da instrução privilegiada que foi capturada pela VMM. Se $i(S_1) = S_2$ significa que o estado S_1 está mapeado no estado S_2 pela instrução *i*, então concordamos que $ij(S_1) = S_2$ significa que existe um estado S_3 tal que $i(S_1) = S_3$ e $j(S_3) = S_2$.

Usando v_i para representar uma série de instruções, então para representar uma série de rotinas de interpretação nós temos $\{v_i\}$, *i = 1 até m,* onde *m* é o número de instruções privilegiadas. Assim como as instruções privilegiadas, o *Dispatcher* e o Alocador também são seqüências de instruções. Um Programa de Controle é então definido como sendo $CP = <D, A, \{v_i\}>$ em função dos seus três módulos. Qualquer programa de controle de interesse para nós serão aqueles que satisfizerem algumas condições. O Programa de Controle em um modelo clássico irá executar em modo supervisor, assim, o PSW no endereço 1 é carregado por *hardware* quando um *trap*

ocorrer, seu modo é ajustado para supervisor e o contador de programa ajustado para a primeira posição do Despachante, iniciando a MV. Observamos também que todos os demais programas executam em modo usuário. Assim, o PSW, que o programa de controle carrega como sua última operação devolvendo o controle de execução do sistema de volta para o programa em operação, terá o seu modo de operação ajustado para usuário.

Neste modelo, a VMM executa em modo supervisor, fazendo com que todos os outros programas tenham execução em modo usuário. Todas as instruções em modo supervisor dos programas são, portanto, interpretadas pela VMM. Desta forma, a VMM exige a alocação de uma área de memória usada para registrar as operações do modo simulado da MV, uma característica que persiste mesmo nas VMM mais recentes.

2.10. Propriedades das MVs

Existem três propriedades de interesse para qualquer programa executando sobre o programa de controle residente (RUTH, et al. 2005):

1. **Eficiência:** Todas as instruções consideradas inócuas (ou seja, que não exigem um *trap*) são executadas diretamente por *hardware* sem intervenção por parte do programa de controle;
2. **Controle de Recursos:** Deve ser impossível para a aplicação rodando sobre o Programa de Controle alterar qualquer recurso do sistema, como por exemplo, a quantidade de memória alocada.
3. **Equivalência:** Qualquer programa *K* executando sobre um programa de controle residente, com duas possíveis exceções, executa de forma indistinguível caso estivesse rodando sem o programa de controle e *K* possui liberdade para acessar qualquer instrução privilegiada.

As únicas duas exceções são caracterizadas por problemas de disponibilidade de recursos e temporização. Em função de alguma penalização sobre o tempo original de execução das instruções de *K* induzidas pelo Programa de Controle, uma seqüência de instruções em *K* pode demorar mais do que o previsto para ser executada, assim, qualquer suposição sobre o tempo de execução de um dado conjunto de instruções pode ser inválida sob determinadas condições. Problemas com disponibilidade de recursos

ocorrem quando o Alocador não consegue atender a uma requisição de alocação de memória, por exemplo, alterando o valor do registrado *R*. Esta situação pode ocorrer uma vez que uma MV é logicamente igual ao *hardware*, mas oferece menos recursos disponíveis para as aplicações. Assim, qualquer VMM deve atender a estes três requisitos de forma que qualquer programa rodando sobre a VMM enxergue uma MV. Assim sendo, o Teorema 1 (GOLDBERG, 1973) define:

> *Teorema 1: Para qualquer computador convencional de terceira geração, um monitor de máquina virtual pode ser construído se o conjunto de instruções sensíveis para este computador é um subconjunto do conjunto de instruções privilegiadas.*

Ao considerar a VMM como um Programa de Controle capaz de manter o homomorfismo (que em álgebra, é uma aplicação que preserva uma estrutura) sobre um conjunto de estados de máquina possíveis *C*, então teríamos *Cv* contendo todos os estados para as quais a VMM está presente na memória e o valor de *P* na PSW armazenada na posição 1 é igual a primeira posição da VMM. O segundo conjunto *Cr* contém os demais estados reais. Os dois conjuntos refletem os possíveis estados da máquina real com e sem a VMM, respectivamente. Cada instrução no conjunto de instruções do processador pode ser pensada como uma operação unária sobre o conjunto de estados $i(S_j) = S_k$. Assim, cada seqüência de instruções pode ser pensada como um operador unário no estado *C*. Sobre este conjunto de instruções, um mapeamento pode ser verificado de forma a atender aos requisitos de uma MV.

Um Mapeamento da Máquina Virtual (VM Map) $f: Cr \rightarrow Cv$ é um homomorfismo um para um para todos os operadores e_i em uma seqüência de instruções *I,* assim, para cada estado $Si \in Cr$ e para qualquer seqüência e_i de instruções, existe uma seqüência e_i' tal que $f(e_i(S_i) = e_i'(f(S_i))$. Esta correspondência é mostrada na Figura 6, relacionando a seqüência de instruções para um conjunto de estados *Cr* real e um conjunto de estados *Cv* virtual.

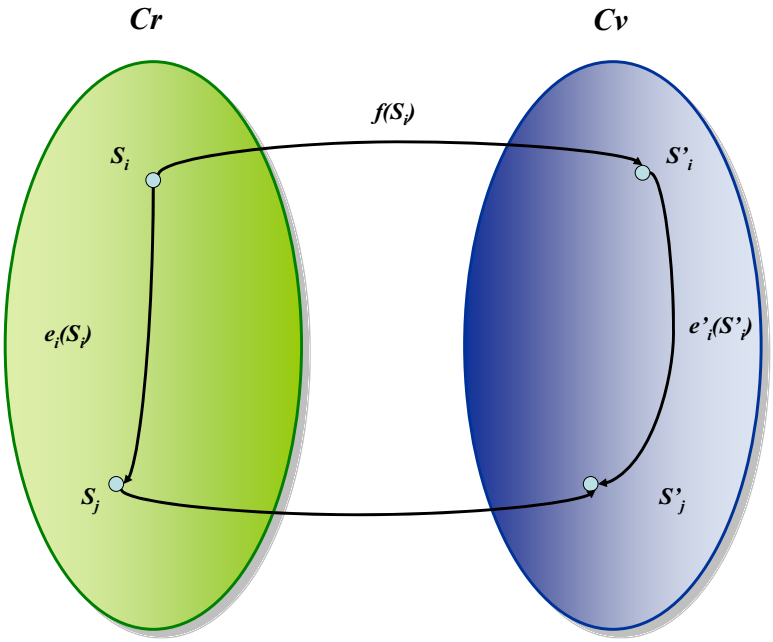

Figura 6: O Mapeamento da Máquina Virtual (GOLDBERG, 1973)

Existe, portanto, um mapeamento matemático dos estados *Cv* da máquina virtual sobre os estados *Cr* da máquina real. Além disso, vemos a existência de uma seqüência de instruções e'_i no domínio dos estados virtuais *Cv* que corresponde à seqüência e_i no domínio dos estados reais *Cr*, o que indica que para cada seqüência de instruções reais e_i existe também uma seqüência de instruções virtuais e'_i equivalente.

Esta definição da Figura 6 também garante a equivalência de funcionamento entre a máquina real e a virtual, onde todas as seqüências de instruções presentes no *hardware* virtual refletem em uma operação similar realizada no *hardware* real. Como apresentado no Teorema 1, um Programa de Controle construído com base nestes fundamentos demonstra as propriedades de equivalência, controle de recursos e eficiência.

Pela definição de instruções sensíveis mostrada anteriormente, é possível definir que apenas instruções inócuas (que não são consideradas instruções sensíveis) podem executar diretamente sobre o processador. Todas as instruções sensíveis sofrem *trap* e são executadas por uma rotina de tratamento de instruções que executam em modo privilegiado ou podem alterar o estado de alocação dos recursos.

Desta forma, a eficiência também é contemplada nas VMMs. A equivalência também é facilmente demonstrada através da Figura 6. Através da aplicação do mapeamento da máquina virtual, todas as instruções inócuas são executadas diretamente sobre o processador sob o conjunto de instruções reais do processador. Todas as

instruções que realizam *trap* são executadas sob um mapeamento similar, entretanto através de rotinas de tratamento do *trap*. Considerando estas características, é possível estabelecer quais as condições necessárias para que uma máquina seja virtualizável, definindo todas as características desejáveis. Em diversos aspectos, a arquitetura x86 não é a rigor totalmente virtualizável, mas a flexibilidade do modelo *trap-and-emulate* permite que uma arquitetura seja virtualizável mesmo não atendendo plenamente aos requisitos do Teorema 1.

2.11. Máquinas Virtuais Híbridas

Poucas máquinas de terceira geração foram projetadas para serem virtualizadas. Um relaxamento nas definições de uma máquina virtual permite a criação de modelos híbridos, que embora não ofereçam a mesma classe de desempenho, oferecem ainda todas as possibilidades oferecidas pela virtualização e podemos chamar de *Máquinas Virtuais Híbridas*. Para simplificar a qualificação dos modelos híbridos, é possível agrupar as instruções sensíveis em dois conjuntos, sem necessariamente dividi-las em duas novas categorias (GOLDBERG, 1973):

- Uma instrução *i* é dita sensível para o usuário se existe um estado *S* = <*E, u, P, R*> para o qual *i* é sensível ao controle ou sensível ao comportamento. Ou seja, mesmo aplicações executando em modo usuário oferecem execução de instruções que são sensíveis ao controle ou a localização.
- Uma instrução *i* é dita sensível para o modo supervisor se existe um estado *S* = <*E, s, P, R*> para o qual *i* é sensível ao controle ou ao comportamento.

Ou seja, tanto o modo usuário quanto o modo supervisor apresentam instruções que do ponto de vista da virtualização apresentam a necessidade de execução através de uma rotina de tratamento para instruções sensíveis e não podem ser executadas diretamente pela MV no processador. Com isso, Goldberg estabelece um segundo teorema (GOLDBERG, 1973):

Teorema 2: Um monitor híbrido de máquina virtual pode ser construído a partir de uma máquina de terceira geração na qual o conjunto de instruções sensíveis para o usuário é um subconjunto do conjunto de instruções privilegiadas.

Assim, para pontuar a principal diferença entre uma MV Híbrida (ou HVM) e uma VMM podemos considerar que em um monitor de HVM, todas as instruções no modo supervisor virtual são interpretadas. Com exceção deste detalhe, todas as demais características de um HVM são iguais as de um VMM. Assim, todas as instruções sensíveis são interceptadas pelo HVM e simuladas.

Em um computador com uma instrução hipotética JSRT 1, cuja função é retornar para o modo usuário, e supondo que esta instrução não é privilegiada, apenas sensível a comportamento, então em função desta instrução um computador não poderia abrigar uma VMM. Entretanto, usando o mecanismo híbrido com interpretação de todas as instruções sensíveis, é possível realizar a virtualização de acordo com os Teoremas 1 e 2 apresentados.

O principal objetivo do mecanismo de virtualização proposto por Goldberg é diminuir qualquer forma de degradação do desempenho devida à execução de um VMM. Com sua proposta baseada no mecanismo de *trap-and-emulate*, um sistema virtualizado ofereceria o máximo de proximidade em termos de funcionamento lógico e aparente ao de um sistema real sem a necessidade de uma camada de interpretação muito extensa e ineficiente, mesmo em casos como o de uma MV Híbrida.

2.12. Tratamento de Instruções Sensíveis

Como observado através dos Teoremas de Goldberg, determinadas características oferecidas por uma ISA não necessariamente atendem aos requisitos de um sistema totalmente virtualizável. Esta deficiência foi contornada por máquinas virtuais híbridas sem, no entanto, conseguir atender plenamente a propriedade de eficiência. Para ser eficiente, uma VMM em arquiteturas não virtualizáveis deve tratar corretamente a execução de instruções sensíveis (WHITAKER, et al. 2005).

Uma técnica para tratar instruções sensíveis é conhecida como *patching*, também normalmente usada em recompilação dinâmica. As instruções críticas são descobertas e analisadas em tempo de execução e recolocadas com uma *trap* dentro da VMM. Este

processo é pouco eficiente e para minimizar os problemas de desempenho, mecanismos para *cache* do código a ser emulado pelo *trap* ou mesmo assistência em *hardware* tem sido proposta, como a tecnologia Intel Virtualization VT-x e VT-i (UHLIG, et al. 2005).

Uma outra abordagem, porém sem exigir mecanismos especiais de virtualização na arquitetura x86 é a da paravirtualização, que nem sempre é atraente em determinadas situações por exigir alterações no Sistema Operacional Hospedeiro (SOH), embora ofereça a menor degradação possível de desempenho.

2.13. Virtualização Clássica no Contexto da Arquitetura x86

Historicamente os sistemas x86 sofreram da falta de suporte para virtualização (ROBIN & IRVINE, 2000), o que exige que a VMM explore técnicas como *patching* que deterioram o desempenho. Outras técnicas baseadas em paravirtualização, aplicadas em máquinas virtuais como o XEN (MENON, et al. 2003), exigem a adaptação do sistema operacional para funcionarem corretamente, o que, embora seja uma das técnicas que permite o melhor desempenho, nem sempre é a melhor alternativa em função das necessidades da solução (BARHAM *et al*, 2000).

Para promover a virtualização clássica do tipo *trap-and-emulate*, Goldberg e Popek definiram parâmetros para medir a capacidade de uma dada ISA de oferecer suporte para a virtualização. De acordo com Adams e Agesen (ADAMS & AGESEN, 2006), uma confusão entre "arquitetura virtualizável" e "arquitetura favorável a execução de *trap-and-emulate*" tem sido feita. Para evitar a confusão entre os dois conceitos, Adams e Agesen sugerem o termo "classicamente virtualizável" para definir as arquiteturas que podem ser virtualizadas totalmente com a adoção de *trap-and-emulate*. Neste caso, a arquitetura x86 não é classicamente virtualizável nos critérios adotados por Popek e Goldberg.

Em uma VMM clássica, o sistema operacional convidado é executado em um nível de privilégio inferior ao da VMM. A VMM é capaz de interceptar todas as instruções que executam leitura ou escrita em áreas protegidas (ou privilegiadas) de contexto. Exemplos destas instruções são as instruções de escrita em uma área de memória mapeada de um dispositivo de E/S. Uma VMM clássica executa o SO diretamente sobre o processador e intercepta as instruções privilegiadas, causando um *trap* e então emulando a sua execução através do mecanismo de execução da VMM,

retornando a resposta da execução para a chamada de instrução do sistema operacional. Esse mecanismo é possível, pois a VMM é executada em um nível de maior prioridade do que o do sistema operacional e esta técnica, explorada por Popek e Goldberg, é conhecida como “desprivilegização”.

2.14. Estruturas Primárias e 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.

Exemplos 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 E/S através de DMA podem alterar a PTE tornando a estrutura sombreada do SO convidado inconsistente com a estrutura primária (BREY, 2003).

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 (SMITH & NAIR, 2005). 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 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. Em um sistema x86, para proteger o sistema hospedeiro do SO convidado, a VMM constrói algumas estruturas de sombra, como uma tabela de páginas de sombra sobre a qual ficam alocadas as tabelas de páginas de endereçamento de 2, 3 ou 4 níveis. O ponteiro que armazena o endereço desta tabela fica em uma área protegida dos registradores.

Quando a VMM passa a gerenciar a tabela de páginas sombreadas, ela trata esta tabela como uma *cache* das tabelas de páginas do SO convidado (ADAMS & AGESEN, 2006). Conforme o SO convidado acessa áreas ainda não alocadas do endereçamento virtual, falhas de paginação geradas por *hardware* são vetorizadas pelo controle da VMM. A VMM então identifica falhas reais de paginação, causadas pelos mecanismos de codificação da política de escrita e leitura da PTE do SO convidado. As falhas reais são encaminhadas para o mecanismo primário de PTE, que cria as novas áreas de endereçamento e seu resultado é então copiado na PTE da área de sombreamento da VMM, que reconstrói seu sombreamento de forma a se manter consistente.

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 mecanismo oneroso 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.

2.15. Otimizações no modelo Clássico de Virtualização

O modelo proposto por Popek e Goldberg pode ser, como dito, relaxado para que a relação VMM e SO possa oferecer uma troca mais favorável de informações que privilegie o desempenho. Desta possibilidade, algumas abordagens com foco na otimização de desempenho surgiram com base em modificações, tanto no *hardware*

quanto no SO, para que o a VMM pudesse oferecer mais flexibilidade na execução virtual.

Uma abordagem é explorar a flexibilidade na interface SO e VMM, o que implica em modificações no SO convidado de forma a oferecer um maior nível de informações para a VMM (KARGER *et al,* 1990). Esta abordagem vive um novo momento sob o nome de paravirtualização.

A outra abordagem é explorar a flexibilidade de *hardware* para que seja possível otimizar o desempenho da VMM. A arquitetura IBM System 370 introduziu a *execução interpretativa*, um modo de execução de *hardware* especial para sistemas operacionais convidados (adicionando um modo de execução a mais no modelo minimalista proposto por Goldberg, como modo usuário e modo supervisor). A VMM executa a maioria das instruções privilegiadas do SO convidado diretamente sobre o modo especial de operação do *hardware*, que prevê um formato específico de operação que é conhecida pela VMM, mas não necessariamente pelo SO convidado (OSISEK, 1991). Esta abordagem garantiu uma significativa diminuição no número de *traps* que precisavam ser processados por uma VMM sem assistência alguma, mesmo que ainda assim alguns *traps* ainda fossem necessários.

2.16. Obstáculos da Arquitetura x86 para a Virtualização

Mesmo ignorando sistemas x86 legados que utilizam os modos x86 "real" e "virtual", os modos protegidos de 32 e 64-bits mais recentes também não são virtualizáveis classicamente e algumas definições da arquitetura x86 oferecem obstáculos:

1. **Visibilidade do Estado de Privilégio:** O sistema convidado pode saber que sofreu uma diminuição no seu privilégio de execução ao ler o seu registrador %cs seletor de segmento, uma vez que o seu nível atual de privilégio (CPL) é armazenado nos dois bits menos significativos de %cs.
2. **Falta de *traps* para instruções privilegiadas que são executadas em nível de usuário:** Instruções privilegiadas como popf ("pop flags") podem alterar as flags da Unidade Lógica Aritmética e das flags de sistema (por exemplo, a Flag IF, que controla as interrupções de sistema). Para um convidado sem privilégios

de sistema, um popf do *kernel* precisa ser emulado através de *trap* pela VMM contra o IF virtual. Infelizmente, um popf sem privilégios não gera alterações no registrador IF, que controla a notificação de interrupção. Desta forma, o *trap* não acontece.

3. **Instruções sensíveis aos Registradores:** São instruções que podem ler ou alterar o conteúdo de registradores e/ou posições de memória tais como registradores de interrupção. As instruções são:

    - SGDT, SIDT, SLDT
    - SMSW
    - PUSHF, POPF

4. **Instruções sensíveis a Proteção do Sistema:** Fazem referência ao sistema de proteção de memória ou sistema de realocação de endereçamento:

    - LAR, LSL, VERR, VERW
    - POP
    - PUSH
    - CALL, JMP, INT n, RET
    - STR
    - MOVE

Embora seja possível listar pelo menos 17 instruções sensíveis (BREY, 2003) e definir algumas restrições ao modelo de virtualização clássica de Popek e Goldberg, basta apenas uma única restrição para que um modelo híbrido tenha que ser adotado para permitir a virtualização da arquitetura x86.

2.17. Considerações Finais

A virtualização de *hardware* por meio de Máquinas Virtuais de Sistema Clássicas, como definido por Popek e Goldberg, estabelece critérios de desempenho, fidelidade e segurança para o funcionamento de VMMs, garantindo, portanto, que uma MV seja capaz de representar fielmente o sistema real do computador hospedeiro. As VMMs clássica são, portanto, o modelo de virtualização capaz de oferecer os recursos de gerenciamento adequados ao funcionamento do *middleware* de virtualização do FlexLab. Entretanto, arquiteturas como a x86 oferecem desafios a virtualização clássica conforme definido por Popek e Goldberg em função da existência de instruções sensíveis a virtualização, exigindo que as MVs adotem um modelo híbrido de

virtualização que combina tradução binária dinâmica (ALTMAN, et al. 1999) com execução por *trap-and-emulate* das instruções sensíveis, mantendo-se ainda dentro dos critérios que estabelecem o funcionamento de uma VMM clássica.

3. COMPARAÇÃO ENTRE AS MÁQUINAS VIRTUAIS

3.1. Considerações Iniciais

As MVs híbridas permitem que computadores dotados de processadores com a arquitetura x86 sejam virtualizáveis por meio dos mecanismos de emulação, tradução binária e tradução binária dinâmica e tratamento de instruções sensíveis por *trap-and-emulate*. Embora sigam os critérios definidos por Popek e Goldberg, as MVs apresentam diferentes abordagens e características de funcionamento, conforme veremos neste Capítulo.

3.2. Tradução Binária na VMM VMware™

As implicações listadas em 2.10 para a virtualização clássica exigem um modelo onde o sistema operacional executa sobre um interpretador ao invés de diretamente sobre o *hardware* físico. Um interpretador pode proteger o vazamento do estado de privilégio atual e mesmo executar corretamente uma instrução popf ao fazer referência a uma CPL virtual ao invés de uma CPL real (durante a execução do *trap*, a instrução é totalmente emulada pela VMM, e não é executada pela CPU real). De certa forma, o papel do interpretador é separar o estado virtual (VCPU) do estado físico (CPU). Entretanto, a interpretação atende apenas a dois dos três requisitos de uma VMM de Popek: Enquanto fidelidade e segurança são alcançados com a interpretação, o desempenho fica muito aquém do que definido na virtualização clássica. Ou seja, o ciclo "carregar-decodificar-executar" pode ser até centenas de vezes mais lenta do que a execução das instruções diretamente sobre o processador (ADAMS & AGESEN, 2006).

Para atender ao terceiro requisito de Popek e Goldberg, a técnica de tradução binária pode ser utilizada para manter o rigor semântico das instruções, porém com alto desempenho. Esta técnica, utilizada na VMM VMware Workstation (WALDSPURGER

, 2003) e também no emulador de alto desempenho QEMU (embora apenas a KQEMU seja considerado uma VMM), permite que a VMM seja construída sobre uma arquitetura não classicamente virtualizável, no caso x86, mas que a VMM seja de fato uma VMM de acordo com a definição clássica. Outros exemplos de tradução binária podem ser encontrados em máquinas virtuais como a Java, que utiliza a tecnologia JIT – Just in time de execução através de tradução binária.

Tomando como exemplo a VMM VMware Workstation, o mecanismo de tradução binária tem como propriedades:

1. **Entrada Binária:** A entrada é em código binário x86, não código fonte.
2. **Dinamicidade:** A tradução ocorre em tempo de execução intercalada com a execução do código gerado
3. **Sob demanda:** O código é traduzido somente quando está para ser executado.
4. **Em Nível de Sistema:** O tradutor não faz inferências sobre o código do sistema convidado. As regras são definidas pela ISA x86, não pela ABI. É diferente de um tradutor em nível de aplicação que pode inferir que "toda chamada a uma função retorna um endereço". A VMM não pode fazer essa inferência, pois precisa manter fidelidade a arquitetura.
5. **Saída é um Subconjunto:** A entrada do tradutor é o conjunto completo de instruções x86, incluindo todas as instruções privilegiadas; a saída é um subconjunto seguro, em geral apenas as instruções do modo usuário (e que são executadas sobre o processador real).
6. **Adaptativa:** O código traduzido é ajustado conforme o formato de execução do sistema operacional convidado para aprimorar a eficiência de execução.

Após testes observados por Adams e Agesen (ADAMS & AGESEN, 2006), a maioria do código de usuário de aplicações executadas sobre a maioria dos sistemas operacionais dispensa o uso da tradução binária como forma de proteger o sistema da execução de instruções não diretamente virtualizáveis, pois estas instruções são raras mesmo em código supervisor do *kernel*. Ao trocar a execução do código usuário entre o mecanismo de execução direta e o mecanismo de tradução binária, nenhum prejuízo foi percebido em termos de execução além da eliminação do tempo necessário pelo tradutor binário. De fato, apenas o código de *kernel* utiliza o mecanismo de tradução binária na VMM VMware em função das instruções sensíveis que são executadas. O código

usuário, que utiliza o processador a maior parte do tempo, é executado diretamente sobre a CPU real.

Outro dado interessante diz respeito à arquitetura de CPUs modernas, como a dos processadores Intel Pentium IV. As CPUs modernas possuem sistemas onerosos de *trap*, desta forma uma VMM que utiliza Tradução Binária pode executar com melhor desempenho instruções privilegiadas do que uma VMM clássica ao evitar o *trap* destas instruções. Os testes conduzidos por Adams e Agesen em CPUs Pentium IV mostraram que três implementações para a execução da instrução privilegiada *rdstc* possuem o seguinte perfil: *Trap-and-emulate* toma 2030 ciclos, *callout-and-emulate* toma 1254 ciclos e emulação na *Cache* de Tradução toma apenas 216 ciclos (utilizando o mecanismo de Tradução Binária). Entretanto, enquanto o *trap* de instruções privilegiadas pode ser totalmente eliminado com a TB, uma fonte ainda mais comum de *traps* ainda se mantém: o *trap* de instruções não privilegiadas que acessam áreas protegidas de dados, como as instruções de *load and store*.

A TB Dinâmica é a técnica utilizada para mitigar este último tipo de *trap*. A idéia básica é a de "inocente até que provem a culpa" (BELLARD, 2003). As instruções do sistema convidado começam a execução como inocentes, garantindo assim que sejam totalmente executadas por traduções idênticas entre o código gerado pelo sistema convidado e o código gerado pelo mecanismo de TB. Durante a execução do código traduzido, a VMM detecta as instruções que sofrem *trap* constantemente e adapta a sua tradução, evitando o *trap* através de uma nova função interpretada ou traduzindo novamente a instrução para evitar o *trap*.

Os mecanismos de Interpretação e Tradução Binária conjugados permitem que grande parte das instruções sejam executadas diretamente pelo processador real, o que garante a fidelidade e segurança de execução sem implicar em uma degradação de desempenho que comprometa a terceira definição de Popek e Goldberg para VMMs: desempenho. Como conseqüência, a paravirtualização não precisa ser utilizada para se alcançar resultados próximos de desempenho sem a necessidade de se adaptar o sistema operacional convidado e hospedeiro.

3.3. Suporte a Virtualização em Hardware

Lançamentos recentes por parte de Intel (Intel Virtualization VT-x e VT-i) e AMD (AMD SVM) permitem que os novos processadores x86 com suporte a estas tecnologias sejam classicamente virtualizados (INTEL VIRTUALIZATION TECHNOLOGY, 2006), já que a similaridade entre ambas as tecnologias são evidentes.

O *hardware* passa a exportar agora uma série de extensões que oferecem suporte a uma VMM clássica para a arquitetura x86. Uma estrutura de dados em memória, chamada de Bloco de Controle da Máquina Virtual, ou VMCB, combina o estado de controle com o subconjunto de estado de uma CPU virtual. Assim, é introduzido também um novo modo de execução, chamado de modo convidado, ou *guest mode*.

O estado de execução em modo convidado possui menos privilégios que o modo supervisor, porém permite a execução direta do código convidado, inclusive de instruções privilegiadas. O antigo ambiente de execução x86 pode ser chamado de *host mode*, ou modo de execução do hospedeiro, sendo este nome motivado pela nova instrução *vmrun,* que muda o modo de execução de *host* para *guest.*

Sobre a execução de *vmrun*, o *hardware* carrega o estado do sistema convidado da VMCB e continua a sua execução em modo *guest*. A execução em modo convidado prossegue até que alguma condição, expressada pela VMM através dos bits de controle da VMCB, é alcançada. Neste momento, o *hardware* realiza uma operação de saída, ou seja, uma operação inversa a executada pela instrução *vmrun*.

Durante a saída, o *hardware* salva o estado do convidado para a VMCB, e carrega o estado fornecido pela VMM dentro do *hardware*, restaurando-o em modo hospedeiro, e então executando a VMM. Os campos de diagnóstico dentro do VMCB auxiliam a VMM a tratar as condições de saída. Após emular o efeito da operação de saída no VMCB, a VMM executa novamente a instrução *vmrun*, retornando para o modo convidado. Os bits de controle do VMCB também oferecem alguma flexibilidade quanto a segurança de execução, permitindo que o sistema operacional convidado assuma controle e o tratamento de interrupções dos periféricos e tabela de páginas.

As recentes extensões de virtualização oferecem uma solução quase completa de virtualização, pois, embora ofereça instruções e estruturas próprias para a troca de controle entre o sistema convidado e o sistema hospedeiro, as novas tecnologias para os processadores x86 ainda não oferecem uma solução explícita para a virtualização da

MMU. Desta forma, todo o processo de rastreamento da memória e estruturas de sombreamento ainda precisam ser implementadas como descrito anteriormente.

3.4. Discussão sobre as extensões de Virtualização para os processadores x86

As extensões introduzidas por Intel e AMD para virtualização permitem que os processadores x86 sejam virtualizados de acordo com a definição clássica. O desempenho depende basicamente da freqüência de operações de saída. Um convidado que nunca provoca saídas para a emulação em modo hospedeiro executa a velocidade nativa, com uma degradação praticamente nula no desempenho geral. Entretanto, se o convidado precisar realizar operações de E/S, então este cenário deve ser reconsiderado, para quaisquer operações de E/S.

Para cada instrução do convidado que disparar uma saída (do modo *guest* para o modo *host* de execução) o tempo de execução será dominado pelas trocas de *hardware* entre os estados. Portanto, reduzir ao máximo o número de saídas é a chave para a otimização das VMMs clássicas sobre a arquitetura x86 com extensões para virtualização.

Para evitar excessivas operações de saída, o suporte a virtualização do x86 implementa idéias similares as idéias introduzidas pelo IBM System 360 de execução interpretativa. Sempre que possível, as instruções privilegiadas afetam o estado da CPU virtual conforme representado dentro do VMCB, ao invés de ocasionarem um *trap* incondicional (UHLIG *et al*, 2005).

Considerando novamente o exemplo da instrução *popf*, uma implementação simplória das extensões x86 poderia disparar saídas para o modo hospedeiro sempre que o sistema convidado executasse uma chamada a instrução popf, permitindo que a VMM atualize os estados virtuais dos bits de interrupção. Entretanto, o sistema convidado pode executar popf com muita freqüência, o que tornaria esta implementação inadmissível em termos de desempenho.

Ao invés de implementar um mecanismo altamente dependente das estruturas de emulação da VMM, as extensões de virtualização do x86 utilizam o VMCB, que inclui uma estrutura de sombra mantida pelo *hardware* dos registradores *%eflags* convidado.

Quando está executando em modo convidado, as instruções operando em %eflags alteram a área sombreada, eliminando a necessidade de saídas desnecessárias.

3.5. Uma comparação qualitativa

A Tradução Binária tende a ser favorecida nas seguintes situações (BELLARD, 2005):

1. **Eliminação de *Traps*:** A TB adaptativa pode substituir a maioria dos *traps* por chamadas a rotinas mais rápidas.
2. **Velocidade de Emulação:** Uma rotina de tratamento pode oferecer a uma rotina de emulação uma instrução pré-decodificada do convidado, já a VMM por *hardware* precisa carregar e decodificar a instrução *trap* do convidado para então emulá-la e propagar a resposta.
3. **Uso reduzido de Chamadas de Tratamento:** Para casos freqüentes observados pela VMM, o mecanismo de TB pode usar as rotinas de emulação da *cache* do tradutor para evitar o custo de chamada das rotinas externas de tratamento.

Da mesma forma, a virtualização por *hardware* é mais favorecida nas seguintes áreas (UHLIG, et al. 2005):

1. **Densidade de Código:** Como não há tradução, a densidade de código é mantida. Na TB, o código é segmentado em blocos básicos e seu tamanho pode ser incrementado para suportar a segmentação dos blocos de tradução.
2. **Precisão de Exceções:** A TB necessita de um trabalho adicional para restaurar o estado do sistema convidado para o código que gera interrupções ou falhas.
3. **Chamadas de Sistema:** Dispensam o uso da VMM

As VMMs por *software* oferecem mais opções para garantir uma execução eficiente do código convidado, entretanto, exigem muito cuidado em sua engenharia e em geral oferecem melhor desempenho para situações onde o acesso a operações de E/S são freqüentes. As VMMs por *hardware* oferecem boas soluções para a virtualização

tradicional por *trap-and-emulate* e por isso, o atual *hardware* ainda não é suficientemente completo para oferecer uma boa solução de virtualização considerando uma solução de *software*/*hardware* para virtualização. Em situações com excessivas condições de saída, a VMM por *hardware* oferece uma considerável degradação de desempenho.

Os gráficos das Figuras 7 e 8 mostram que as diferenças de desempenho entre a execução da VMM por software e por hardware são bastante próximas, entretanto, fica também claro que um dos principais problemas de desempenho para a VMM por hardware é o tratamento de interrupções gerado pela MMU. Talvez esta seja uma das razões pela qual seu desempenho no comparativo não tenha sido superior frente à VMM por software, embora ambas tenham oferecido um desempenho para aplicações de computação intensiva próximo do tempo de execução nativa e um desempenho para aplicações com operações complexas de acesso a memória com degradação de desempenho frente a execução nativa.

O comparativo mostrado por Adams e Agensen também mostra que a VMM por software oferece melhor execução virtualizada de operações relativas a MMU, enquanto que a VMM por hardware é melhor no tratamento de chamadas de sistema). Uma possibilidade é utilizar dinamicamente as duas técnicas dentro de uma mesma máquina virtual, onde através de heurísticas uma ou outra técnica é dinamicamente acionada. O benefício possível, de acordo com as Figuras 7 e 8, é uma melhor execução frente ao código nativo, o que representaria por volta de 4% a 5% de degradação de desempenho para aplicações de computação intensiva e 15% a 20% de degradação de desempenho para aplicações com operações complexas de memória, chamadas de sistema e E/S.

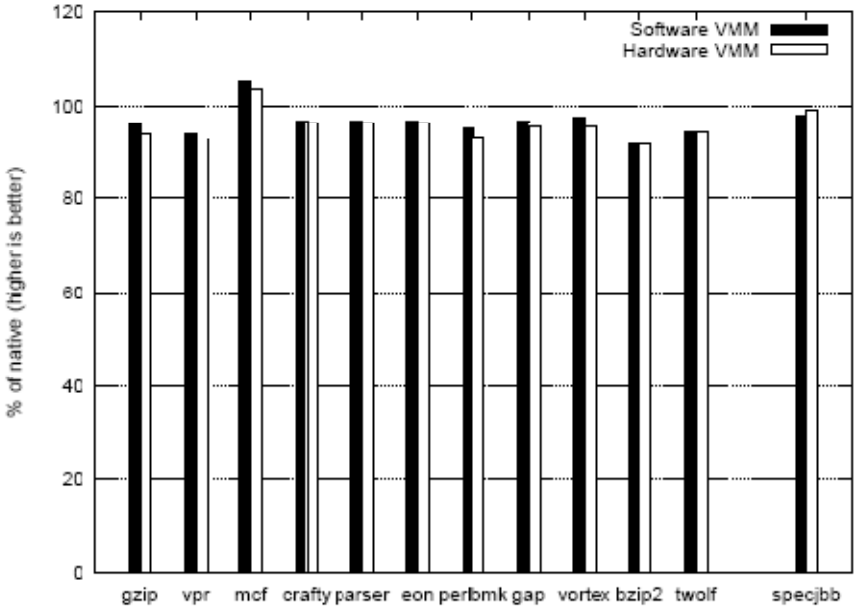

Figura 7: Testes comparativos entre a VMM por software e a VMM por hardware utilizando SPECint 2000 e SPECjbb 2005. Quanto maiores forem as barras melhores são os resultados (ADAMS & AGESEN, 2006).

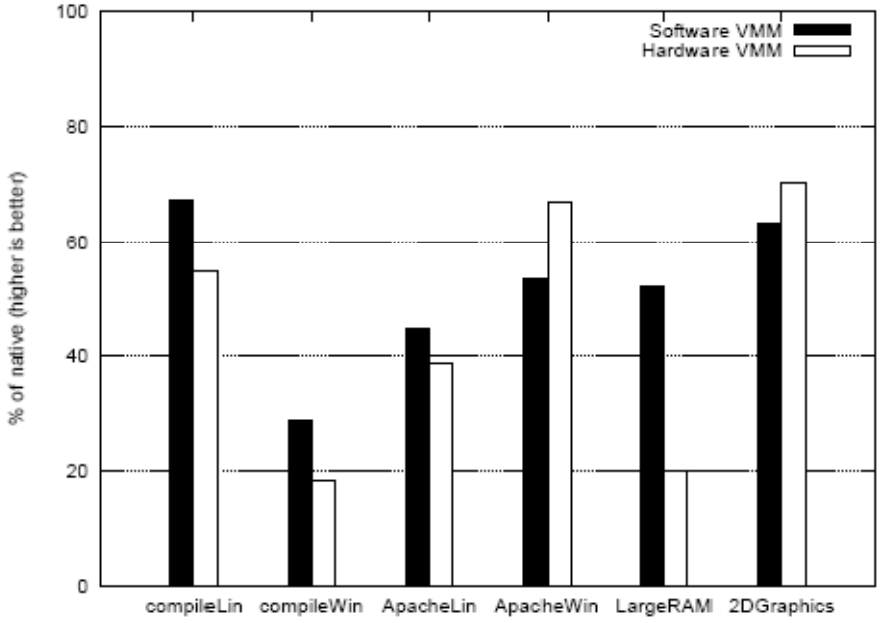

Figura 8: Avaliações utilizando aplicações com diversas operações de E/S. Quanto maiores forem as barras melhores são os resultados (ADAMS & AGESEN, 2006).

3.6. Semelhanças entre a VMM VMware e o KQEMU

O projeto QEMU (BELLARD, 2005) utiliza o mesmo conceito de Tradução Binária implementado na VMM por *software* VMware utilizando um mecanismo de tradução dinâmica de código aberto, rápido e maduro. No atual estágio, o QEMU é um

emulador completo para diversas arquiteturas (incluindo x86, PowerPC, ARM, MIPS e SPARC) e que permite que um computador seja executado dentro de outro. Por usar emulação, a degradação de desempenho em relação a execução nativa é na ordem de 4 a 10 vezes para a execução do Windows XP emulado sobre a QEMU de um computador x86, respectivamente para operações com inteiros e ponto flutuante (MENON, et al, 2005).

Como utiliza emulação completa, a QEMU permite que um sistema operacional possa executar sobre outro computador sem exigir nenhuma alteração, e oferece também uma oferta de dispositivos genéricos de E/S emulados:

- Display VGA
- 16450 portas seriais
- Mouse P/S 2
- Disco Rígido por interface emulada
- Placa de Rede NE2000
- Dispositivos Genéricos
- *Debugger*
- Interface administrativa de usuário

A tradução dinâmica do QEMU executa uma conversão em tempo de execução das instruções da CPU emulada para as instruções da CPU do sistema hospedeiro. O resultado é um código binário armazenado na *cache* de tradução (TC) de forma que possa ser reutilizado. A vantagem para um interpretador, assim como já visto no VMware, é que o código só é procurado *(fetch)*e decodificado *(decode)* apenas uma vez.

Para oferecer melhor desempenho, a TC possui 16MB de espaço para armazenar as instruções mais recentes. Entretanto, o desempenho do emulador QEMU não permite pretensões mais ambiciosas, e em função das definições de Popek e Goldberg para uma VMM, a QEMU não pode ser considerada uma VMM, embora garanta um ambiente seguro e isolado de execução. Nesta condição, a QEMU se difere da VMM VMware justamente por não permitir que a maioria das instruções do sistema convidado sejam executadas diretamente pelo processador do *hardware* real. Essa limitação é contornada com uma versão especial da QEMU dotada de um acelerador, chamada de KQEMU (BELLARD, 2005).

A KQEMU, ao contrário do QEMU, não possui o código fonte aberto, mas é distribuída gratuitamente. Basicamente implementa um mecanismo de *trap-and-emulate* onde a maioria das instruções de *kernel* e instruções não virtualizáveis (já vistas anteriormente) são emuladas usando o mecanismo de tradução binária dinâmica e suas respostas são propagadas para o sistema convidado. Como a grande maioria das instruções do sistema convidado é inócua (ou seja, não provocam ou não são instruções sensíveis, de acordo com a definição de Goldberg), podem ser executadas diretamente pelo processador real.

3.7. Paravirtualização

Sistemas de virtualização como o projeto XEN (BARHMAN, 2004) e o projeto Denali (WHITAKER, SHAW & GRIBBLE, 2002) procuram diminuir a degradação de desempenho causada pelos mecanismos de virtualização usando a técnica de paravirtualização. Esta abordagem oferece uma forte isolação entre as diferentes instâncias de MVs.

O desempenho de um sistema paravirtualizado é bastante próximo do desempenho nativo do computador real, entretanto, os sistemas paravirtualizados exigem que o sistema operacional convidado seja modificado para ser capaz de trocar informações específicas com a VMM, possibilitando assim a paravirtualização.

Algumas instruções são adicionadas e outras são removidas da MV, e, como algumas instruções x86 são deixadas de fora da MV, mecanismos como a tradução binária, não são mais necessárias para oferecer uma arquitetura virtualizável classicamente. Como a arquitetura x86 é bastante complexa, a remoção de algumas instruções simplifica a virtualização através da VMM.

Na VMM Denali, o suporte ao sistema de E/S é feito em nível de ISA, assim, por exemplo, um frame Ethernet inteiro pode ser enviado para a VMM com apenas uma instrução. Esta forma singular de acrescentar ou diminuir o número de instruções da MV reduz o número de *traps* do SO convidado contra a VMM. Assim, no sistema Denali, o sistema operacional convidado é executado diretamente sobre o *hardware*, sem a necessidade de um SO hospedeiro. A VMM também implementa uma camada completa de programas de controle dos recursos de *hardware*, o que permite um melhor

isolamento e uso dos recursos de *hardware*, mas torna complexa a implementação da VMM.

Assim como o mecanismo de carregamento de processos é diferente e exige modificações no SO convidado, o tratamento de interrupções também é feito através da VMM, que cria uma fila de interrupções entre todas as MVs e trata as interrupções direcionadas a uma máquina quando esta entra em operação. Esta abordagem reduz bastante o número de trocas de contexto e conseqüentemente, a degradação de desempenho. Para suportar um bom desempenho, os sistemas paravirtualizados sacrificam inclusive a total compatibilidade com a arquitetura x86.

3.8. Considerações Finais

Os mecanismos de virtualização estão consolidados em três grandes grupos dentro da arquitetura x86: Tradução Binária Dinâmica, Virtualização Assistida por *Hardware* e Paravirtualização (ou virtualização assistida por Sistema Operacional). Dentro do conceito de virtualização distribuída proposto pelo projeto FlexLab e explicado em detalhes no Capítulo 5, a virtualização através da Tradução Binária Dinâmica apresenta um bom compromisso entre requisitos de *hardware* e *software* e desempenho.

A virtualização na arquitetura x86 é um benefício difícil de ser alcançado sem que haja ou uma VMM complexa, capaz de tratar todas as exceções provocadas pelas inúmeras situações onde instruções x86 não são virtualizáveis, ou assistência de *hardware* ou sistema operacional. Nestes dois últimos casos, existe o inconveniente da solução adotada não funcionar em todos os computadores do parque atual de máquinas, uma vez que exigem ou processadores específicos ou portes específicos do sistema operacional, duas características indesejáveis para uma solução como o FlexLab.

4. SISTEMAS DE VIRTUALIZAÇÃO DISTRIBUÍDA

4.1. Considerações Iniciais

A possibilidade de distribuir o processamento e acesso aos dados em uma rede local levou à criação de ambientes distribuídos conhecidos como *clusters* computacionais. Um *cluster* computacional pode ser definido como um conjunto fracamente acoplado de computadores independentes, agrupado para a realização de uma tarefa computacional, visando oferecer uma imagem de sistema única (*SSI – Single System Image*), conforme a Figura 9:

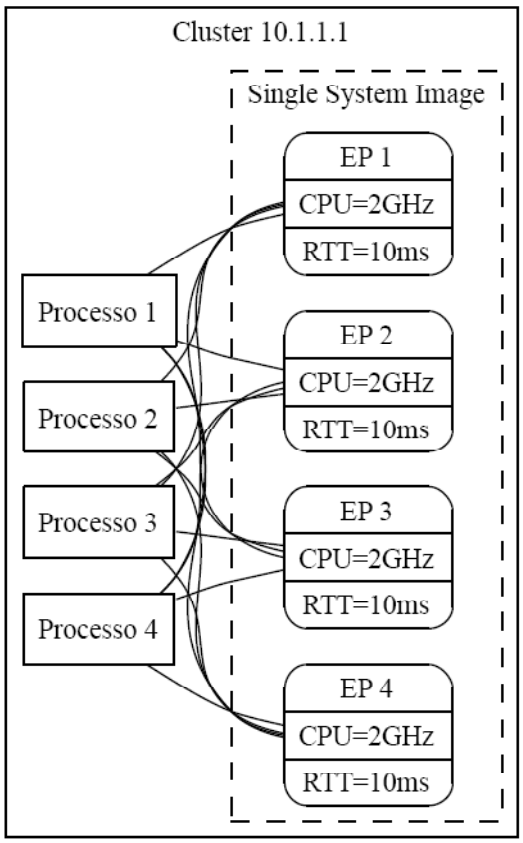

Figura 9: Arquitetura de um *Cluster* (DODONOV, 2006).

Clusters são, geralmente, compostos por computadores homogêneos interconectados por meio de uma rede local de alto desempenho.

Com o aumento contínuo da complexidade das tarefas a serem solucionadas e com o alto custo de recursos especializados, tais como máquinas paralelas e *clusters*, a necessidade de uma tecnologia que possa utilizar recursos distribuídos em escala superior tornou-se evidente.

Essa nova tecnologia foi denominada de *grid computing*, ou *grids computacionais* (FOSTER, et al. 2001). *Grids* são baseadas na suposição de que a maior parte dos computadores conectados à Internet passam grande parte do tempo ociosos, sendo possível utilizar uma parcela do seu tempo ocioso para executar tarefas distribuídas.

4.2. Motivações para o desenvolvimento de Sistemas de Processamento Distribuído

A conexão dos *grids computacionais* com a Internet é utilizada para a distribuição transparente de tarefas como mostra a Figura 10, aumentando significativamente a capacidade de processamento do ambiente, sem nenhuma restrição teórica quanto ao número e arquitetura de estações participantes do sistema. Entretanto, a tecnologia de *grids* enfrenta diversos problemas como:

1. **Configuração heterogênea das estações** – diferentemente dos sistemas distribuídos baseados em *clusters* e máquinas paralelas, as estações presentes num *grid* computacional não apresentam configuração homogênea, podendo variar uma série de parâmetros de configuração, desde a arquitetura do sistema e dispositivos presentes até mesmo no sistema operacional utilizado e classe de tarefas executáveis;

2. **Possibilidade de operações desconectadas** – uma vez que a tecnologia de *grids* utiliza o tempo ocioso de estações conectadas à Internet, há a possibilidade que sejam desligadas, ou utilizadas por aplicações locais do proprietário. Dessa forma, o processamento realizado na estação pode ser perdido caso não seja migrado;

3. **Magnitude da distribuição** – diferentemente de *clusters* e máquinas paralelas, em *grids* computacionais o número de estações é variável. Dessa forma, não é possível ter uma imagem única do sistema a não ser através da adoção de tecnologias de virtualização sob condições específicas;

4. **Variação de latência** – a variação de latência de acessos aos dados é altamente dependente do ambiente utilizado, alterando em função dos protocolos de comunicação, amplitude de distribuição de dados, tamanho de mensagens entre outros fatores (DODONOV, et al. 2005).

O resultado é a ocorrência de variações não-lineares de tempos de acesso e taxas de transferência de dados entre *EPs* de um mesmo ambiente distribuído. Como é observado, *grids* computacionais devem tratar questões inerentes à tecnologia, tais como a heterogeneidade, operações desconectadas e escalabilidade das estações, as quais incluem a segurança das operações e conexões, distribuição de processamento, balanceamento de carga e variação de latência.

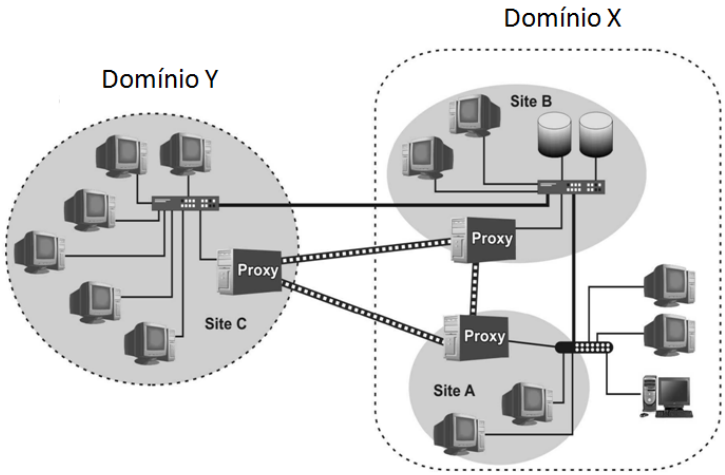

Figura 10: Arquitetura de um *Grid* heterogêneo (DODONOV, et al. 2005)

4.3. Ambientes Distribuídos Virtuais

Diversos projetos têm explorado a utilização de um ambiente distribuído virtual para aplicações de *Grid* (FIGUEIREDO, et al. 2003) como forma de explorar a flexibilidade e gerenciamento aprimorado do processamento distribuído através de nós virtualizados com baixa degradação de desempenho. Além disso, graças a capacidade

de isolamento entre diferentes instâncias, as MVs têm sido empregadas em *Grids* para oferecer um sistema mais flexível de utilização permitindo mais de uma máquina virtual por nó sem perder a consistência de isolamento entre as MVs concorrentes.

Os ambientes de *Grids* virtualizados permitem a migração do estado de execução de cada máquina virtual abrindo espaço para mecanismos de balanceamento de carga (ZHAO, et al. 2006) e criação e destruição dinâmica de nós de processamento baseados em virtualização (ZHANG, et al. 2004).

Em geral, a necessidade de uma ambiente de computação em *Grid* que ofereça eficiente compartilhamento e distribuição de recursos, flexibilidade, segurança e isolamento, abrem espaço para a adoção de MVs como a base para a oferta de compartilhamento de recursos, segurança e principalmente, administração de recursos (FIGUEIREDO et al. 2003). Desta forma, diversos *middlewares* para computação em *Grid* foram propostos, como o VMPlants (KRSUL, e al. 2004), PlanetLab, In-VIGO (ABDALA, et al. 2005), Virtuoso, entre outros. Um ambiente distribuído virtual propõe:

- Flexibilidade de Configuração;
- Escalabilidade;
- Rápida iniciação da MV;
- Tolerância a Falhas;
- Interoperabilidade;
- Gerenciamento de Recursos.

O ambiente VMPlants procura atender a todos os requisitos listados acima, e ainda permite que:

- O usuário defina o seu ambiente de execução virtual;
- Arquivar, copiar e compartilhar seu ambiente de execução virtual;
- Instanciar diversos clones de um sistema virtualizado;
- Salvar o estado de uma execução atual;
- Migrar o estado de uma execução atual;
- Retomar a execução de um estado salvo.

A utilização de MVs, como no projeto VMPlants, permite uma administração simplificada dos serviços oferecidos no *Grid*, uma vez que cada ambiente de execução pode ser customizado e clonado para ser instanciado em *n* nós de operação concorrentemente. A Figura 11 mostra o processo simplificado de configuração de um ambiente distribuído virtual utilizando o *middleware* VMPlants.

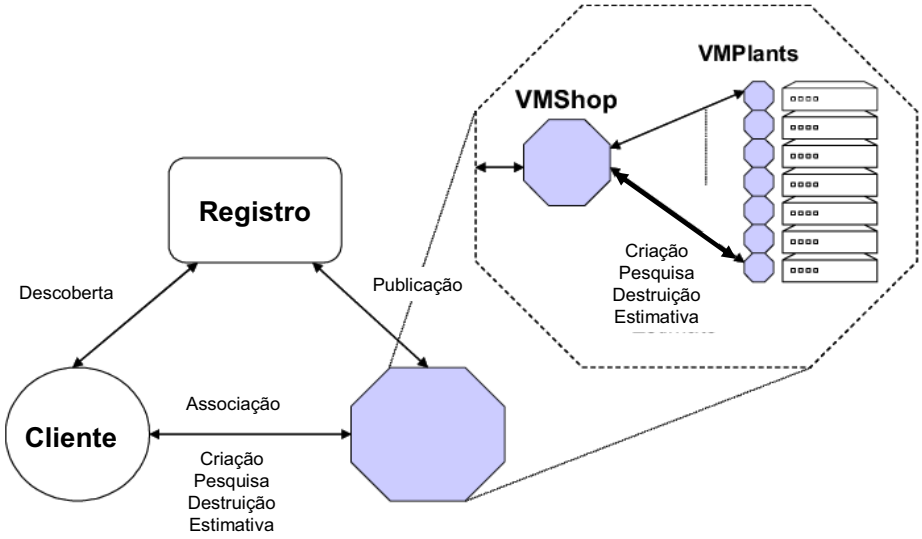

Figura 11: Arquitetura do *middleware* VMPlants (FIGUEREDO, et al. 2003)

De acordo com a arquitetura do VMPlants, um cliente pode pesquisar ou requisitar a criação de uma MV personalizada com base em um imagem padrão (*Golden Imagem)*, e então associar essa nova imagem a uma configuração do *Grid* (ou iniciar no *Grid* uma configuração já salva, caso a imagem já esteja configurada). O serviço VMShop se encarrega de comunicar os nós do ambiente distribuído e criar, destruir ou simplesmente levantar estimativas de uso de cada máquina virtual operacional no ambiente. Desta forma, a criação de uma MV é feita através de um comando enviado pelo VMShop para um ou *x* nós do ambiente VMPlants.

Este ambiente copia a imagem da máquina virtual para o seu *hardware* e inicia a execução da máquina virtual. Através do *middleware*, uma máquina virtual pode ser salva (seu estado de execução) ou destruída, de acordo com os scripts de execução desenvolvidos para o gerenciamento do ambiente VMPlants (scripts que comandam o início e o final de execução de uma MV específica funcionando localmente ao nó).

A infra-estrutura de virtualização é flexível e pode ser oferecida por produtos de mercado como o VMware ESX Server e o VMware Workstation e o desempenho de

execução de cada nó é compatível com um tempo de execução virtualizado, já explorado no item 3.6.

Em experimento conduzido pelo projeto VMPlants (KRSUL, et al. 2004), foi criado um ambiente com nós configurados com Servidores Dual Pentium 4 2.4GHz 1.5GB RAM, 18GB SCSI e suporte a MVs VMware GSX 2.5.1, além de um servidor Pentium 3 1.8 GHz, 1GB RAM, 500GB SCSI RAID 5 acessado através de NFS como servidor de armazenamento dos discos virtuais utilizados pelo VMPlants. Cada nó se conecta ao servidor de imagens através de um switch Gigabit Ethernet utilizando placas de rede Ethernet 100Mbps.

Cada nó é configurado com uma VM VMware dotada de interface de rede e as mensagens de criação são enviadas pelo servidor VMShop. Foi criada uma seqüência de testes com três tamanhos de imagens: 32MB, 64MB e 256MB. Para cada tamanho de imagem foram feitas 128, 128 e 40 requisições, respectivamente, onde cada requisição significa a criação e iniciação de uma MV nos nós. A Figura 12 mostra um gráfico normalizado com os tempos de criação de uma MV de ponta-a-ponta, onde pode se observar o tempo gasto para a criação bem sucedida para cada configuração de MV.

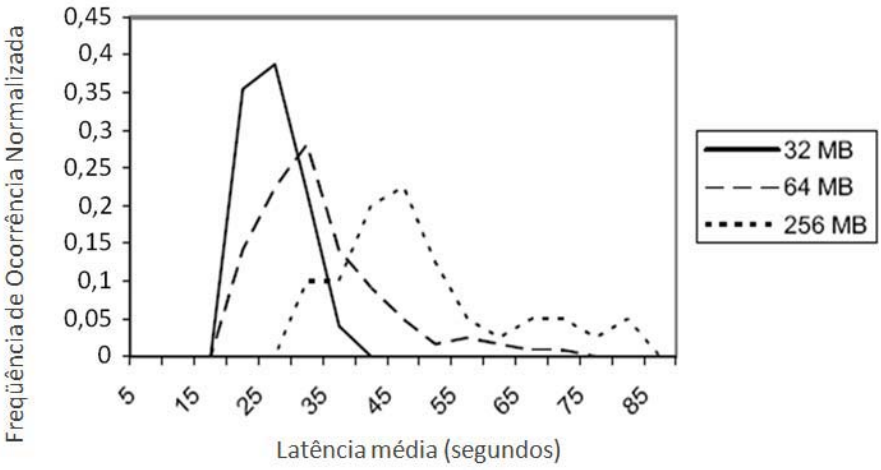

Figura 12: Distribuição normalizada da latência para a criação de imagens medidas entre o ponto de criação de uma MV e sua iniciação nos nós (KRSUL, et al. 2004).

A MV VMware, assim como outras MVs como a KQEMU ou VirtualPC, permitem, inclusive através de APIs, salvar o estado atual de execução da imagem (estados dos registradores internos, imagem de RAMDISK, processos, etc) em um arquivo binário para que a execução da imagem virtual possa ser retomada posteriormente por outras MVs, inclusive. O *middleware* VMPlants oferece este recurso através de scripts de controle entre o servidor VMShop e os nós de virtualização. A

Figura 13 mostra o gráfico dos tempos de clonagem de cada uma das configurações de MVs a partir de uma única *Golden Image*. Para imagens pequenas com 32MB, o tempo médio de clonagem é de 10 segundos.

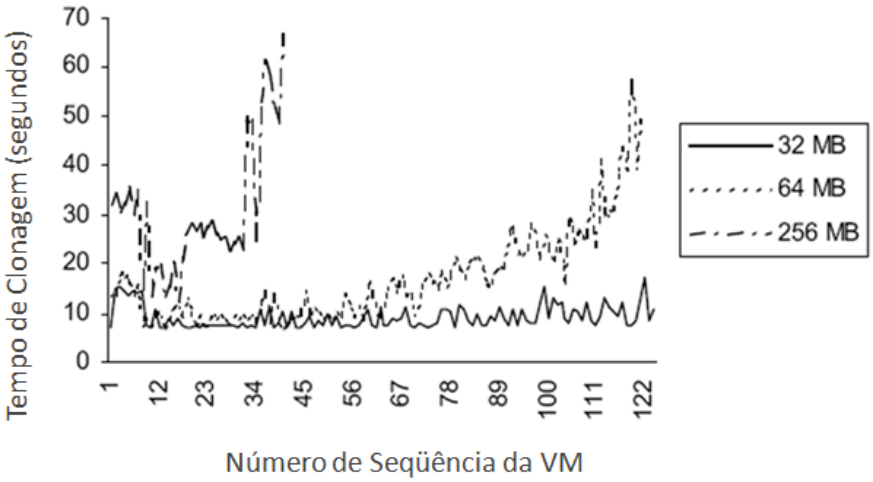

Figura 13: Tempo de clonagem para um número total de requisições seqüências para cada tipo de configuração de imagem (KRSUL, et al. 2004).

4.4. Virtualização de Discos

O gerenciamento de computadores de forma geral tem se tornado cada vez mais complexo à medida que o acoplamento entre *hardware*, *software* e dados se torna mais forte, muito em função da evolução de todos os sistemas e subsistemas computacionais nos últimos anos (ZHOU, ZHANG & XIE, 2006). Em um contexto de computadores em rede, o desafio de gerenciar os computadores sob uma perspectiva individual e de ambiente de rede se torna ainda maior, com diversas dificuldades introduzidas à medida que os computadores passam a armazenar localmente grande quantidade de dados.

Uma alternativa a este problema é a utilização de um gerenciamento centralizado com discos virtuais distribuídos entre as estações e armazenados em um repositório central de discos virtuais. Este paradigma prevê que o computador no nó de gerenciamento não precisa utilizar um disco rígido local, e todo o processo de iniciação e utilização de dados possa ser feito através de *streaming* de *software* dos dados contidos no disco virtual e armazenado no repositório central de imagens.

Esta abordagem é diferente da utilizada em *thin-clients*, pois ao contrário desta solução, todo o processamento é realizado no computador local, desta forma, a solução

aproveita todo o poder computacional local e permite que o servidor de imagens virtuais seja mais simples e barato.

Tecnologias de disco virtuais distribuídos têm sido demonstradas recentemente em forma de soluções de mercado, como o sistema de virtualização de discos da Ardence Disk Services (ARDENCE, 2006) e também projetos acadêmicos como o TransCom (*Transparent Computing*) que implementa um mecanismo de gerenciamento centralizado de discos virtuais (ZHOU et al. 2006). Em ambos os casos, os computadores executam suas tarefas sem um armazenamento local, entretanto, todas as instruções das aplicações são executadas pelos computadores clientes, o que permite uma utilização mais racional dos processadores de baixo custo destes computadores.

A tecnologia chave é a utilização de discos virtuais que simulam um dispositivo de armazenamento físico baseado em blocos usando arquivos de imagens localizados nos repositórios de imagens e acessados através de rede. O sistema TransCom permite:

1. **Suporte a sistemas operacionais heterogêneos:** Com modificações mínimas qualquer sistema operacional pode mapear um disco virtual ao invés do dispositivo real.
2. **Transparência:** Para o usuário e aplicações, a utilização do disco virtual é idêntica ao do disco rígido físico.
3. **Compartilhamento flexível de dados e aplicativos:** O compartilhamento de arquivos e aplicativos é transparente, o que permite uma utilização completa do mecanismo de discos virtuais.

Entre as diversas aplicações, o mecanismo de discos virtuais distribuídos é especialmente interessante para aplicações educacionais (configuração e gerenciamento de salas de aulas), aplicações militares (segurança de dados) e aplicações comerciais corporativas (simplicidade de gerenciamento e redução de custo de *hardware*), além de redução no custo total de propriedade ao eliminar diversos problemas relacionados ao gerenciamento individualizado de cada computador (ZHOU et al. 2006).

A utilização do TransCom, assim como de sistemas similares como o Ardence Disk Services, é baseada na identificação única de cada cliente através do endereço MAC e mostrada na Figura 14. Utilizando o protocolo embarcado nas placas de rede com suporte a PXE, cada cliente é capaz de iniciar o processo de *boot* atribuindo a seção de *boot* ao endereço do *software* embarcado de *boot* executado a partir da ROM da placa

de rede, geralmente baseado no mecanismo de *Etherboot* (BAUM et al. 2001). O papel do *Etherboot* é oferecer uma pré-iniciação do computador cliente, identificando serviços básicos como conectividade de rede e área de memória RAM para criação do RAMDISK.

O *software Etherboot* também identifica um servidor de rede, geralmente implementando o protocolo TFTP, para transferir do servidor central de imagens a imagem correta contendo o disco virtual. O disco virtual é mapeado como um disco real e, portanto, oferece condições para o sistema operacional possa realizar um *bootstrap* e iniciar o processo de rede, mesmo que o arquivo binário inteiro não tenha sido transferido do repositório para o cliente. Esta característica "sob-demanda" é oferecida pelo repositório de imagens, que ofereça um serviço de *streaming* de *software*.

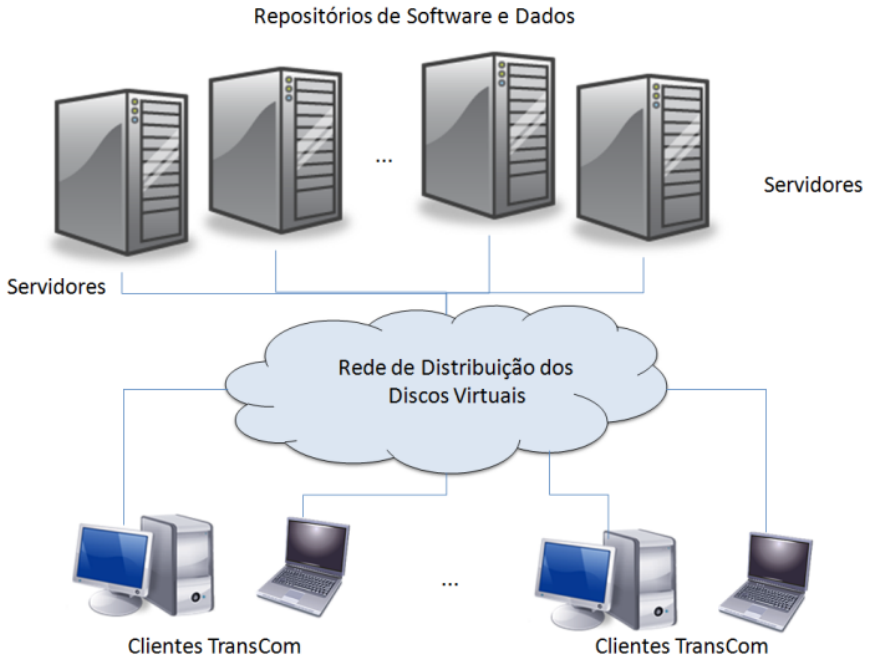

Figura 14: Arquitetura geral de um sistema TransCom de Discos Virtuais Distribuídos (ZHOU, et al. 2006).

Durante o processo de *boot*, o cliente executa através do protocolo PXE a transferência de um sistema de pré-inicialização que oferece as funções do disco virtual.

A partir deste sistema de pré-*boot*, o cliente pode escolher o sistema operacional desejado e iniciar o *boot* através do disco virtual. No sistema operacional, os *drivers* de acesso a disco são substituídos pelo *driver* de disco virtual e a partir deste ponto, todos os acessos a disco serão feitos através do disco virtual, de forma transparente, como se os dados estivessem locais.

Um benefício deste sistema é que, por ser um arquivo emulando um disco virtual, o disco virtual distribuído pode incorporar integralmente as alterações sofridas durante a execução do sistema cliente, ou então pode ignorar todas as alterações, oferecendo, portanto uma opção de gerenciamento que não permite a alteração das imagens. Esta opção oferece diversas vantagens, como maior consistência de funcionamento das imagens (que não podem ser alteradas sem autorização prévia) e segurança (por não permitir o armazenamento local de dados sensíveis).

Os dados de desempenho do projeto TransCom na Tabela 1 mostram que a solução de discos virtuais distribuídos podem substituir um sistema de discos físicos locais. Para tal conclusão, experimentos foram conduzidos em um ambiente composto por 30 clientes com processador Intel Celeron 1GHz, 128MB DDR RAM e placa de rede 100Mbps *on-board* e um servidor de imagens com processador AMD Athlon64 3000+ com 2GB RAM de placa de rede Ethernet 1Gbps, além de um disco rígido de 80GB e 7200 RPM. Tanto clientes quanto servidores estão conectados por um switch Ethernet com 40 portas de 100Mbps e duas portas de 1Gbps (para os servidores).

O servidor utiliza o sistema operacional Microsoft Windows 2003 Server e os clientes utilizam o Microsoft Windows XP. A Tabela 1 mostra um comparativo entre os tempos de iniciação do sistema através de um disco local e através do disco virtual remoto (*boot* remoto).

A métrica de desempenho utilizada é o tempo em segundos, desde o tempo em que a máquina é ligada até o surgimento da tela de *login* do Windows, após concluído o processo de *boot* remoto.

Tabela 1: Latência de *Boot* remoto do sistema TransCom (ZHOU et al. 2006).

	PC	Clientes TransCom		
		1	10	20
Boot do SO	53,13	48,73	70,62	92,79
MS Word 2003	2,23	1,26	2,28	6,35
MS PPT 2003	5,21	3,04	6,58	9,98
Photoshop V7.0	13,29	11,08	16,48	27,51
Flash V 6.0	18,62	7,16	31,41	74,3
3D MAX V4.0	29,71	25,68	34,24	54,18
Cópia de um Arquivo (20 MB)	11,59	8,95	19,75	37,51
Cópia de um Arquivo (50 MB)	28,24	24,33	49,48	109,52

A pior situação de latência é a cópia de um arquivo de 50MB realizada simultaneamente por todas as estações cliente. Neste cenário, a latência é

comparativamente o pior caso em relação ao disco rígido local e em comparação ao processo de *boot*, onde o desempenho é bom em função da localidade de dados no servidor (já que todas as estações clientes utilizaram o mesmo disco virtual).

4.5. O Modelo de Virtualização Distribuída

O modelo de virtualização distribuída oferece um paradigma no qual a virtualização é explorada em diversos nós físicos de processamento, podendo reproduzir de forma simplificada e em escala diversos cenários de configuração e uso para um ambiente de rede pela simples replicação do ambiente virtual em cada nó, sem que isso limite a utilização ou execução de aplicações em cada nó. Esta arquitetura também permite a sua aplicação em diversos modelos de simulação que podem explorar a virtualização distribuída, como nos modelos de avaliação de desempenho e simulação de uma rede de clusters (APOSTOLOPOULOS, et al. 2006).

A virtualização tem sido aplicada em escala comercial na forma de produtos, como o VMware Virtual Desktop Infrastructure (VMWARE, 2007), que criam uma infra-estrutura de *desktops* virtualizados rodando sobre um único servidor e oferecendo o acesso via tecnologia *Remote Desktop* aos seus usuários. Desta forma, todos os benefícios da virtualização são utilizados pelos usuários sem que estes tenham que instalar ou executar seu ambiente virtualizado em seus computadores.

Para os propósitos de gerenciamento, esta abordagem traz alguns riscos, como a dificuldade de dimensionamento de um servidor em função do custo e do número de usuários concorrentes utilizando a infra-estrutura virtual, e também a ausência de gerenciamento na configuração de *software* utilizada pelo computador do usuário para acessar esta infra-estrutura virtual (se o computador do usuário tornar-se indisponível por algum razão, a infra-estrutura virtual de *desktops* torna-se irrelevante). A Figura 15 mostra um exemplo de uma arquitetura de *desktops* virtualizados.

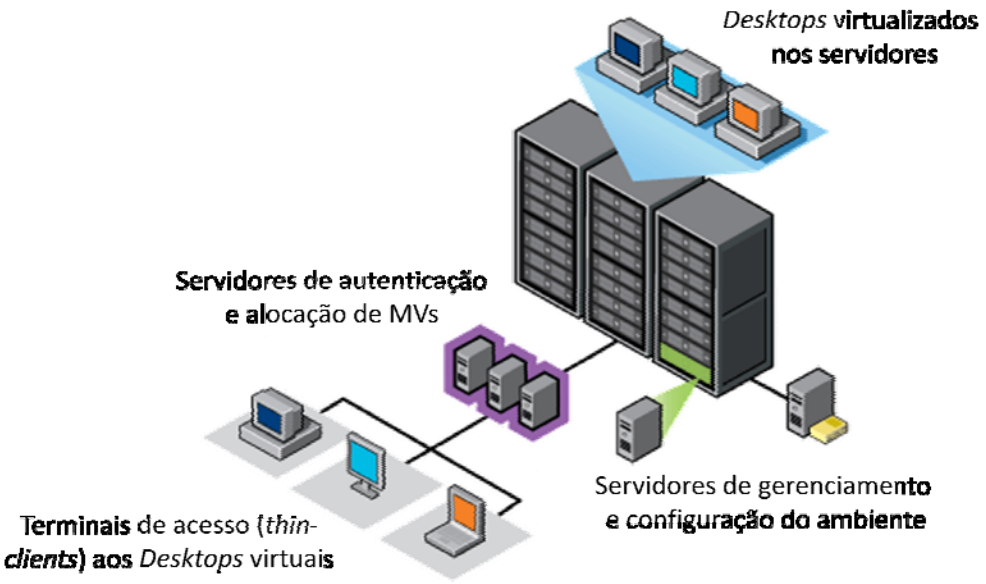

Figura 15: Infra-estrutura virtual de *Desktops* utilizada através de *thin-clients* (VMWARE, 2008).

A possibilidade de utilização da virtualização da infra-estrutura de *Desktops* a partir de um modelo distribuídos representa a quebra do paradigma atual de uso da virtualização e é uma das motivações deste estudo, que procura também explorar as técnicas de virtualização e Sistemas de Arquivos Distribuídos (SADs) combinadas para gerenciamento de computadores em redes locais (FARRELL, et al. 1995).

Modelos de MVs distribuídas já são exploradas cientificamente em aplicações de *Grids* distribuídos. Ivan Krsul e co-autores (KRSUL, et al. 2004) propuseram pela primeira vez o uso de MVs em computação em *Grid*, identificando como vantagens a isolação de ambientes, a segurança, controle de recursos, independência de ambientes de execução e suporte a aplicações legadas. Os projetos In-VIGO e VMPlants estão entre os primeiros a utilizar a virtualização como ferramenta para simplificar a disponibilização de um ambiente compartilhado de processamento e o processo de submissão de trabalhos e modelo de execução de um *Grid*. Em trabalho similar, Paul Ruth e co-autores (RUTH, et al. 2005) argumentam que a virtualização distribuída em um ambiente de computação em *Grid* oferece apenas a isolação entre as MVs. Um conjunto de MVs trabalhando em conjunto em um ambiente de computação compartilhada na Internet, por exemplo, continuam expostas a outros computadores da rede e por isso, propõem a criação de uma rede virtualizada criada através de agentes de virtualização, que foi chamada de Violin. Cada MV está conectada a uma rede virtual única e por isso são inacessíveis por outras redes.

Em ambos os casos, o ambiente de computação em *Grid* trabalha com ISUs com tamanho relativamente pequeno, em ordens de Megabytes, e, portanto, são relativamente simples de serem transportadas para os nós de processamento, mesmo através de redes WAN (ADABALLA, et al. 2005).

O fato dos sistemas de computação em *Grid* virtualizados utilizarem sistemas auto-contidos permite que recursos típicos da virtualização, como a migração de MVs para outros computadores hospedeiros, simplifique o processo de migração de MVs entre nós do *Grid*, desta forma, nós com processamento comprometido podem transferir, mediante agentes, sua MV e seu estado de execução para outro ponto da rede (SAPUNTZAKIS, et al. 2004) de forma otimizada para que esta operação possa se realizar em ambientes de redes WAN.

Embora o FlexLab ofereça um *middleware* com os mesmos recursos de um *middleware* para computação em *Grid* (KANEDA, et al. 2005), seu objetivo principal é oferecer o gerenciamento de computadores através de ISUs em redes LAN, o que torna dispensável a utilização de agentes, como o Violin, para suportar a criação de uma rede virtualizada ou a migração de MVs entre nós da rede, embora isto seja possível.

4.6. Considerações Finais

A evolução das redes locais de computadores e dos mecanismos de distribuição de ambientes de processamento cria novas oportunidades para que Sistemas de Arquivos Distribuídos sejam usados além do suporte a ambientes de *Cluster* e computação em *Grid*, como na criação de ambientes com MVs distribuídas. Ao contrário dos trabalhos atuais envolvendo MVs, onde o propósito de se criar um ambiente utilitário de computação distribuída é a maior motivação do uso da virtualização, abordagens onde MVs são utilizadas para executar Imagens Únicas de Sistema, como no projeto TransCom, que apesar de não utilizar virtualização tira proveito do *streaming* de *software*, mostram que tecnologias para computação em rede estão maduras para este tipo de arquitetura de solução aplicado fora do ambiente de computação *Grid*.

5. ARQUITETURA DO FLEXLAB

5.1. Considerações Iniciais

Neste Capítulo serão apresentados os elementos que compõem a arquitetura do FlexLab e o funcionamento do Sistema de Arquivos multi-camada, que contribui para a otimização do desempenho do *middleware* de virtualização distribuída. Este Capítulo também apresenta um estudo sobre as MVs disponíveis atualmente no mercado e na academia e como elas podem ser utilizadas dentro do modelo de virtualização distribuída como ferramenta para gerenciamento de uma rede de computadores. Por fim, o Capítulo apresenta o funcionamento do FlexLab e suas características de execução.

5.2. Gerenciamento Centralizado de Computadores

Soluções desenvolvidas com a finalidade de oferecer recursos para gerenciamento de computadores em rede adotam arquiteturas para concentrar as operações administrativas em um ponto central, desta forma, todas as políticas e atividades de gerenciamento são acionadas a partir de um único ponto para se refletir nos demais nós gerenciados da rede (CHERVENAK, et al. 2000). A possibilidade de gerenciar computadores de um ponto central se reflete na arquitetura de rede adotada por estas soluções, sendo que a maioria destas arquiteturas apresenta um computador atuando como servidor com serviços de gerenciamento e diversos clientes conectados a ele, enviando informações de *status* e recebendo informações e comandos de configuração, como mostra a Figura 16.

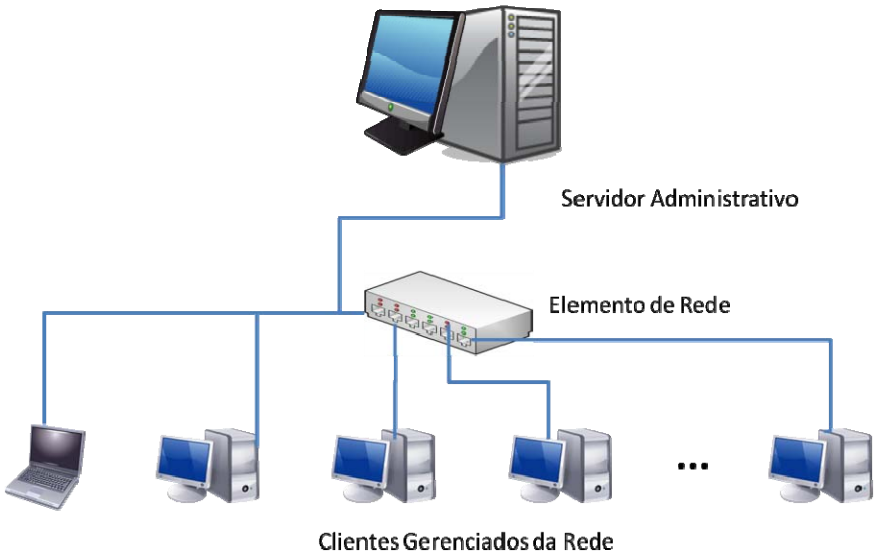

Figura 16: Arquitetura de rede de computadores utilizando um servidor de gerenciamento e clientes gerenciados na rede (MORGADO, CRUZ & TWANI, 2008).

Entre os serviços de gerenciamento oferecidos pelas soluções que tiram proveito da arquitetura de gerenciamento centralizado, destacam-se:

- Definição centralizada dos parâmetros de configuração de *software* para todos os computadores;
- Distribuição, instalação e configuração centralizada de pacotes de *software* aplicativo;
- Definição centralizada de políticas de uso e permissões de acesso aos recursos dos computadores;
- Definições centralizadas dos serviços de segurança, proteção de dados e cópias de segurança de informações;
- Geração centralizada de inventários de *hardware* e *software*.

O que se busca com todos os serviços listados anteriormente é a automação de tarefas repetitivas e relativamente simples de configuração de *software*. Entre os modelos de gerenciamento centralizado de computadores, pode se considerar dois grandes grupos de soluções:

1. **Execução Centralizada:** Soluções onde a execução centralizada dos ambientes de trabalho dos clientes no servidor da rede é a principal característica para se ter um gerenciamento centralizado;

2. **Configuração Centralizada:** Soluções onde a centralização das informações de configuração de *software* dos computadores clientes é a principal característica para se ter um gerenciamento centralizado.

Enquadram-se no Grupo 1 soluções como gerenciamento através de *thin-clients* e infra-estrutura virtual de *desktops* (*Virtual Desktop Infra-structures*). Em ambos os casos, o dimensionamento das configurações de *hardware* do servidor, responsável por hospedar a maior parte ou todo o processamento dos computadores clientes, é o principal gargalo operacional para se obter uma solução que ofereça um custo-benefício linear em função do número de computadores. Por exemplo, o custo para se criar uma infra-estrutura para gerenciar 20 computadores pode não ser compatível com as necessidades da organização, sendo o mesmo válido para projetos com mais de centenas de computadores, uma vez que o número de computadores poderia exigir investimentos em equipamentos de alto desempenho, como os encontrados em *data centers*, para se criar a central de servidores.

No Grupo 2 encontram-se soluções diversas que aplicam seus mecanismos de gerenciamento em camada de aplicativo, tais como *scripts* que automatizam atividades básicas de gerenciamento, e também soluções que trabalham na camada de sistema operacional, usando o conceito de Imagem de Sistema Única (ISU) (ZHOU, ZHANG & XIE. 2006). Neste caso, o mecanismo principal de gerenciamento utiliza uma imagem de um disco rígido, que é um arquivo binário representando os volumes lógicos de um disco, suas partições e dados armazenados, gerada a partir de um computador cuja configuração de *software* é considerada como sendo o padrão a ser replicado nos demais computadores da rede.

Soluções baseadas em gerenciamento através de Imagens de Sistema Únicas, assim como as baseadas em *scripts* em camada de aplicativos, sofrem com as limitações impostas pelo forte acoplamento entre *hardware* e *software* das estações, sendo praticamente impossível gerenciar uma rede de computadores heterogêneos com apenas uma ISU ou um único *script* padrão de configuração.

Em função das inúmeras possibilidades de configuração geradas pela combinação de diferentes configurações de *hardware* e *software*, a complexidade de gerenciamento através de uma ISU ou *script* único acaba se tornando uma tarefa individual e pode inviabilizar estas soluções (SIRER, et al. 2001).

5.3. A Proposta do FlexLab

O objetivo do FlexLab é o desenvolvimento de um *middleware* de virtualização distribuída que tem por finalidade oferecer um mecanismo centralizado para gerenciamento de computadores. Sua proposta é oferecer os benefícios do gerenciamento centralizado de computadores a partir de uma Imagem de Sistema Única armazenada em um ponto central da rede e executada nas MVs distribuídas nos computadores clientes.

O *middleware* do FlexLab propõe a utilização da abstração de *hardware* oferecida pela MV e a capacidade de *streaming* de *software* do Sistema de Arquivos Distribuídos para permitir que uma MV seja capaz de oferecer um ambiente de execução completo ao seus usuários. A execução distribuída do FlexLab permite que o servidor da solução seja usado apenas para oferecer acesso ao disco virtual que a MV usa para criar seu ambiente de execução, uma atividade computacional mais simples do que concentrar a execução de todos aplicativos dos usuários da rede em um único ponto. Desta forma, o FlexLab foi projetado para permitir uma configuração flexível para um ambiente de informática, como um laboratório de informática com estações de pequeno porte ou um *cluster* multi-uso com estações *commodity* e que disponibiliza diversos serviços de processamento (AGUIAR, et al 2008)(b).

Mesmo no caso mais severo em termos de desempenho e recursos de *hardware* com configuração heterogênea, o FlexLab permite que um ou diversos computadores sejam configurados com uma única imagem contendo o mesmo sistema operacional e aplicações, precisando apenas de um conjunto de *hardware* com processador, memória RAM e placa de rede *Ethernet* com o protocolo PXE habilitado para permitir o *boot* pela rede (INTEL, 1999).

Com este conjunto de *hardware*, o computador consegue iniciar o processo de carregamento através da rede e todas as camadas de *software*, incluindo máquina virtual, sistema operacional e aplicações, são oferecidas pelo servidor do FlexLab localizado em um ponto central da rede local. A iniciação da ISU nos clientes e a posterior execução de aplicativos passa a utilizar recursos locais de cada computador cliente, caracterizando assim uma execução distribuída da ISU.

Como tira proveito da abstração de *hardware* promovida pela máquina virtual, detalhes como configuração de *drivers*, variáveis de ambiente e interface com *hardware*

são padronizados para todos os nós de virtualização, o que permite a utilização de computadores com configurações de *hardware* heterogêneas, inclusive dispensando o disco rígido local, uma vez que as ISUs são acessadas através da rede pelo Sistema de Arquivos Distribuídos.

O FlexLab, cuja arquitetura é apresentada na Figura 17, é composto por:

a. Repositório de discos virtuais, cada um sendo uma versão de Imagem de Sistema Única;
b. Um servidor do FlexLab, permitindo o *boot* remoto das ISUs armazenadas;
c. Um ambiente de rede que oferece uma largura de banda condizente com os requisitos de desempenho do laboratório;
d. Os computadores que atuam como nós de virtualização e,
e. Opcionalmente, uma estação de gerenciamento.

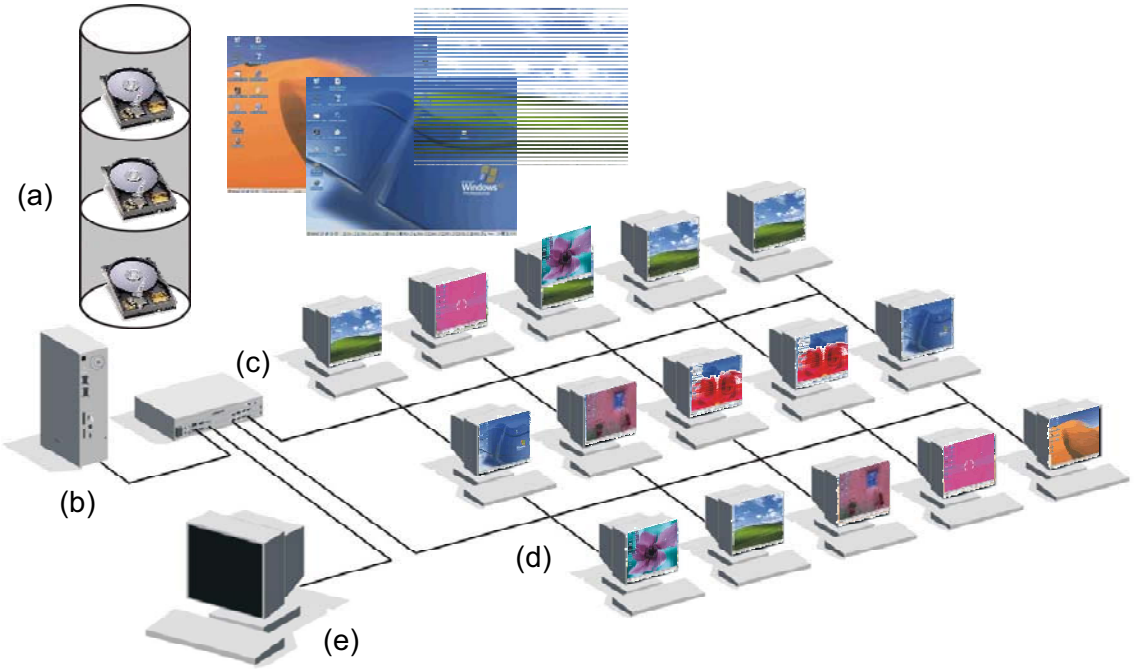

Figura 17: Uma ambiente virtualizado distribuído utilizando o FlexLab.

Em função da sua arquitetura, todos os computadores podem ser iniciados com uma única imagem virtual através de *boot* remoto, embora cada computador mantenha uma instância de execução distinta e independente dos demais computadores. O *boot* remoto

é uma tecnologia desenvolvida pela Intel (HENRY, 2000) com base no protocolo de rede PXE e é um padrão aberto adotado pela indústria desde então para permitir que clientes de uma rede corporativa possam carregar automaticamente através da rede imagens de *software* e configurações. Com o *boot* remoto, as estações dos clientes não necessitam realizar o *boot* pelo disco local, cenário onde os esforços de gerenciamento das imagens são pulverizados entre as estações da rede. Além disso, toda a instalação e manutenção de aplicações são realizadas apenas uma vez para uma única imagem, já que o FlexLab, e sistemas que tiram proveito do protocolo PXE em geral, permite a execução de múltiplas instâncias desta imagem em diversos clientes concorrentemente.

O FlexLab necessita dos seguintes requisitos de *hardware*:

- Ambiente de rede com switch 10/100 mbps;
- Servidor ou servidores em *cluster* com configuração equivalente ao processador Intel Pentium IV 1.8GHz, com 512MB de memória RAM e placa de rede *Ethernet* 100 mbps;
- Computadores clientes com processador arquitetura x86, equivalente ao processador Intel Pentium III com 1 GHz e 256MB de memória RAM. Os computadores clientes exigem placa de rede *Ethernet* 10/100 mbps com suporte ao protocolo de rede PXE.

O funcionamento do FlexLab é orientado a conectividade, uma vez que a ISU é utilizada através de um SAD que se torna indisponível sem a conexão de rede, embora a estação só se torne indisponível em um cenário de falta de conectividade apenas quando o sistema operacional ou aplicativos requisitarem um bloco de dados que não foi ainda requisitado ao servidor do FlexLab. O diagrama da Figura 18 apresenta o processo de *boot* dos computadores da rede. É necessário que cada computador inicie o *boot* a partir da placa de rede (PXE *Boot* LAN).

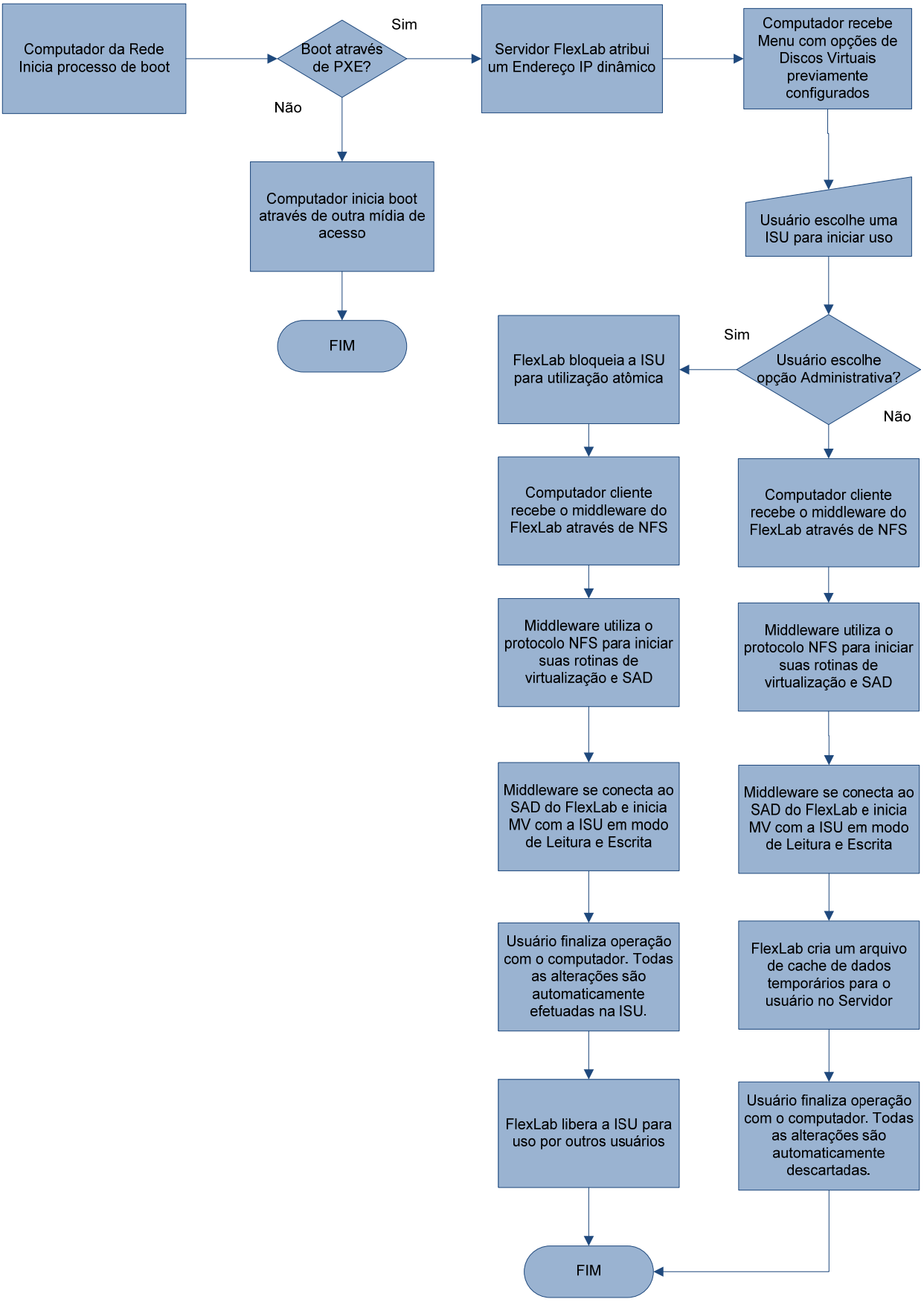

Figura 18: Diagrama de funcionamento do FlexLab sob o ponto de vista dos computadores clientes.

O FlexLab utiliza técnicas de *streaming* de *software* para permitir a execução distribuída de MVs, e em função disto, o FlexLab não pode ser considerada uma solução do Grupo 1, uma vez que todo o processamento é local ao nó virtualizado. Ao contrário das soluções do Grupo 2, o FlexLab não tem dependência do acoplamento entre *hardware* e *software* de um computador, simplificando a utilização de ISUs.

5.4. Recursos do *Middleware* FlexLab

O escopo do projeto do FlexLab define uma série de requisitos que o *middleware* deve oferecer para que ele possa ser aplicado dentro da proposta de uso como ferramenta de gerenciamento de computadores utilizando o conceito ISU. O *middleware* do FlexLab deve:

- Oferecer gerenciamento de redes de computadores com processador de arquitetura x86 compatível com sistemas de virtualização;
- Permitir o *boot* remoto do *middleware* e também da ISU utilizada pela MV em uma única operação;
- Permitir que um arquivo contendo um ISU possa ser acessado concorrentemente por diversas MVs distribuídas;
- Permitir aos administradores da rede que impeçam os usuários de alterarem de forma permanente as configurações de *software* dos computadores;
- Permitir que a camada da MV possa ser alterada dinamicamente durante o tempo de *boot* para que seja escolhida a MV com o mecanismo de virtualização (VMM, Hypervisor, Virtualização Completa) mais adequado para o tipo de processador do computador;
- Permitir que o *kernel* utilizado pelo *middleware* ofereça todos os *drivers* de periféricos de *hardware* para a correta iniciação do computador;
- Permitir que a MV seja executada a partir do próprio servidor para fins administrativos;
- Permitir que as MVs também sejam distribuídas para ambientes em *Grid* e reaproveitamento de *clusters* multi-uso.
- Permitir a utilização de Sistemas Operacionais Convidados (SOC) na MV sem a necessidade de alteração ou recompilação do seu código fonte para

suportar mecanismos específicos de virtualização. Mecanismos de paravirtualização, embora ofereçam desempenho próximo da execução nativa (SUGERMAN, et al. 2001), exigem que o sistema operacional seja recompilado para que possa interpretar corretamente os mecanismos de tratamento de instruções não virtualizáveis da arquitetura x86 (KING & CHEN, 2002). Isto é inviável nos SOs de código-fonte fechado, como o Microsoft Windows ou o Microsoft Virtual PC (MICROSOFT, 2007).

5.5. *Middleware* Virtualizado com Arquitetura Distribuída Multicamada

O *middleware* do FlexLab é composto pelos seguintes elementos principais:

- *Kernel* Linux versão 2.6.17 ou 2.6.21 preparado para *boot* remoto;
- Sistema Operacional Hospedeiro, no caso uma instalação Linux modificada baseada na distribuição Fedora Core 3, oferecendo apenas os recursos e serviços essenciais para o funcionamento da MV;
- MV baseada em QEMU com o módulo de aceleração KQEMU ou KVM.

A Figura 19 apresenta as camadas de *software* executadas no cliente FlexLab. O *middleware* FlexLab, por questões de desempenho, é transportado em uma camada diferente do arquivo de ISU, caracterizando uma Arquitetura Distribuída Multicamada.

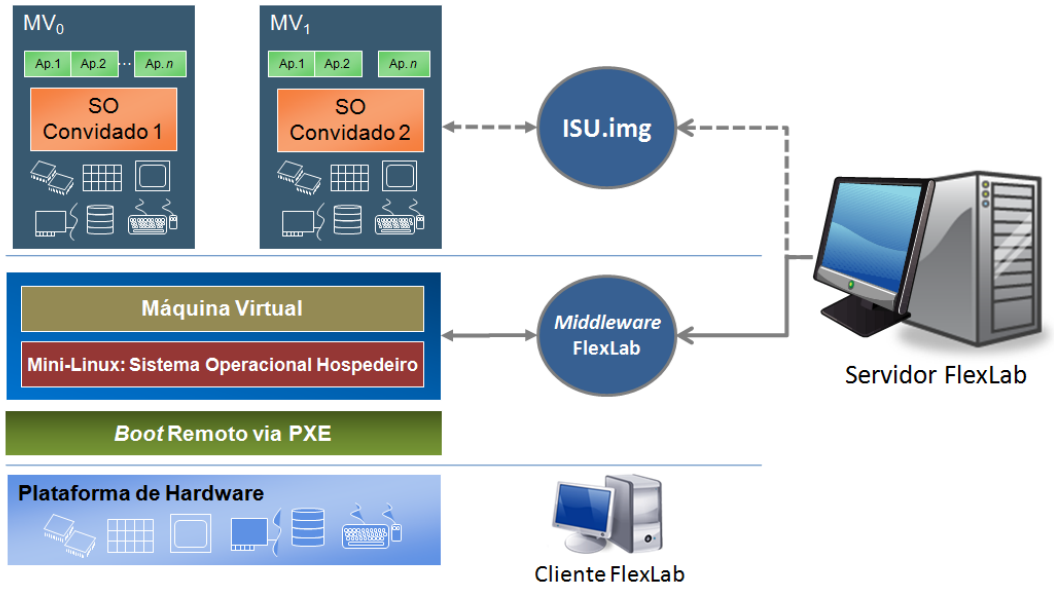

Figura 19: Arquitetura multicamada do *middleware* FlexLab.

A arquitetura multicamada do *middleware* FlexLab não necessita de nenhum *software* instalado no cliente. Todas as camadas de *software* podem ser utilizadas a partir de um servidor remoto, embora o processamento seja sempre local no computador iniciado remotamente.

Embora não seja o foco deste trabalho, a possibilidade de execução de diversas MVs sobre o *middleware* do FlexLab também é possível, como mostra a Figura anterior.

Ao contrário de sistemas de *boot* remoto que utilizam apenas uma camada para oferecer o acesso a ISU (ARDENCE, 2007) (CITRIX, 2008) (ZHOU, 2006) (ZHAO, 2006), o *middleware* do FlexLab utiliza duas camadas:

Camada 1: *Boot* remoto do Sistema Operacional Hospedeiro (SOH), contendo uma versão modificada reduzida da distribuição Linux Fedora Core 3 e que abriga a MV utilizada na camada de abstração da solução. O SOH é reduzido para que possa ser copiado através do sistema de arquivos NFS (SUN MICROSYSTEMS, 1984) versão 3 diretamente do servidor FlexLab para a memória RAM do cliente. Desta forma, o cliente FlexLab dispensa a necessidade de ter um disco rígido físico no computador e o desempenho de execução do *middleware* do FlexLab é aprimorado em função da melhor localidade de dados.

Camada 2: Na camada 2 acontece a requisição de acesso da MV em execução a ISU que está armazenada no servidor FlexLab. Na camada 2, a ISU, um arquivo do tipo *.img* que representa um disco rígido com acesso a bloco de dados com tamanho na ordem de gigabytes, é transferida em forma de *streaming* para o computador cliente (KUACHAROEN, 2004). Neste formato de transferência, apenas os blocos requisitados pela MV são transferidos pela rede. O SOH, de acordo com configuração prévia no servidor FlexLab, monta um Sistema de Arquivos baseado em NFSv4 sobre UDP, com melhor desempenho em redes com poucos computadores (AGUIAR, et al. 2008)(a) ou utiliza um SAD com distribuição de carga baseado em pNFS/PVFS.

O *boot* remoto do *middleware* FlexLab na Camada 1 utiliza a tecnologia SLIM sobre o sistema de arquivos NFSv3. O SLIM é um sistema utilizado em *Clusters* de computadores que operam sem disco rígido e permite a construção de um sistema de

boot remoto usado para que o SOH se inicie no computador cliente. Uma vez iniciado no computador cliente, o SOH consegue iniciar a MV que, através da Camada 2, iniciará a ISU do FlexLab. O SLIM oferece um ótimo desempenho de *boot* com pouco tráfego de rede se comparado a sistemas similares, como o DRBL (YANG, CHEN & CHEN, 2005).

Através do SLIM, os computadores clientes conseguem iniciar o *middleware* FlexLab. Após o *boot* através do PXE, o computador cliente recebe um endereço de IP dinâmico utilizando o protocolo DHCP (*Dynamic Host Configuration Protocol*). Através de um sistema de transferência trivial de dados baseado em TFTP (*Trivial File Transfer Protocol*), o *bootstrap* do *kernel* do Linux é copiado na memória RAM do cliente e, a partir da inicialização do *driver* de rede, cria-se um sistema de arquivos baseado no NFSv3 para o início da cópia em RAM do restante do *middleware* FlexLab baseado em Linux.

A arquitetura do *middleware* baseada em duas camadas oferece vantagens:

- Arquivos da ISU, por trafegarem em uma camada de conexão diferente da camada do SOH, não comprometem a conexão do *middleware* do FlexLab, que é tratada por outro serviço no servidor. Desta forma, o FlexLab é capaz de atender diversas requisições concorrentes de *boot*.
- A arquitetura multicamada permite que o SOH seja utilizado pelo computador cliente através de mais de um mecanismo de cópia. Algumas pastas de sistema que não precisam ser alteradas pelos clientes, como por exemplo, as pastas /usr, /opt, /etc, /sbin e /bin, são utilizadas diretamente do servidor pelo sistema de arquivos NFSv3, economizando espaço na memória RAM dos clientes. Pastas que são modificadas pelos clientes durante o processo de *boot* do SOH, como por exemplo, as pastas /dev e /root, são copiadas diretamente para a memória RAM de cada cliente. Neste caso, utiliza-se o sistema de cópia com sincronismo *rsync* (TRIDGELL & MACKERRAS, 1996), que otimiza o processo de leitura e escrita a arquivos ao copiar na memória RAM do cliente, ou na pasta equivalente do servidor FlexLab, apenas o bloco de dados alterados entre as duas partes, ao invés de copiar todos os blocos de dados de um arquivo em sua totalidade. O mecanismo do algoritmo *rsync* analisa a quantidade de dados alterados, comprime estes dados e somente trafega pela rede a diferença entre as duas

pontas, otimizando o fluxo de dados na rede e as operações de leitura e escrita.

- Como a camada de transporte da ISU é separada, o *middleware* FlexLab pode decidir, ou ser configurado, para mudar o mecanismo de transporte de NFSv3, mais simples de se implantar porém mais suscetível a falhas de transmissão em ambientes com mais de 10 acessos concorrentes, ou utilizar um sistema de arquivos que emprega técnicas de acesso paralelizado com balanceamento de carga, como é o caso do pNFS/PVFS. Estudos (AGUIAR, et al. 2008)(b) mostram que o modelo multicamada utilizando o sistema de arquivos híbridos com balanceamento pNFS/PVFS melhora o desempenho de acesso aos arquivos da ISU.

A Figura 20 mostra o modelo de duas camadas do *middleware* FlexLab. O arquivo contendo a ISU da MV pode ser acessado na Camada 2 pelo(s) computador(es) cliente(s) tanto por NFS quanto pelo SAD usando pNFS/PVFS2, distribuindo e balanceando a carga de acesso com um ou mais servidores.

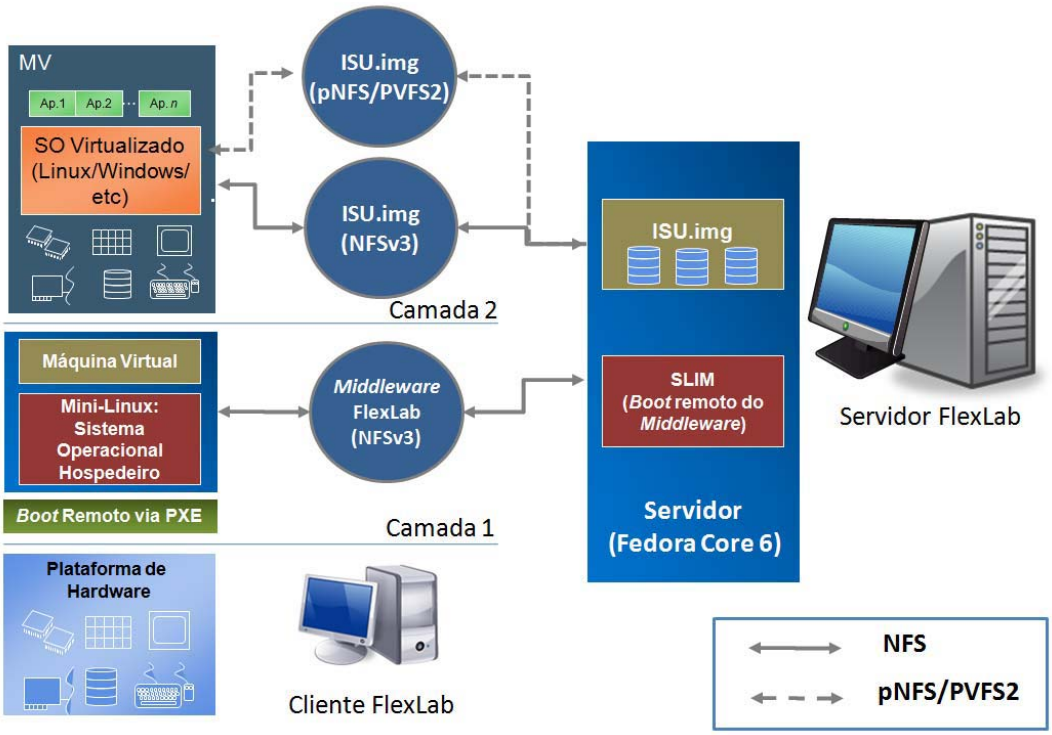

Figura 20: Camadas 1 e 2 do Sistema de Arquivos do FlexLab.

5.6. Arquitetura do Sistema de Arquivos Distribuídos do FlexLab

O projeto FlexLab tem por objetivo desenvolver estudos sobre as oportunidades do *middleware* de virtualização distribuída. Com a aplicação do Sistema de Arquivos Distribuídos no *middleware* FlexLab (AGUIAR, et al. 2008)(a), o desempenho do FlexLab em ambientes com mais de 10 computadores foi aprimorado, principalmente nos cenários de pior caso onde todos os computadores da rede realizam acesso simultâneo a ISU do FlexLab.

O SAD do FlexLab é baseado no protocolo NFSv3, compatível com a maioria dos sistemas operacionais (ANDERSON & CHASE, 2000) e que apresenta diversas aplicações em sistemas de armazenamento e acesso a disco em rede de alta velocidade (SAPUNTZAKIS, 2003). Além disso, o NFSv3 é compatível com o sistema usado pelo FlexLab para balancear a carga das operações de leitura e escrita realizadas pela Camada 2 de transporte da ISU.

Como o *middleware* FlexLab é responsável por distribuir arquivos ISU de grande porte (ordem de gigabytes) para diversas máquinas acessando um ou mais arquivos concorrentemente, adotamos a estratégia de utilizar um SAD capaz de implementar um sistema de arquivos paralelizado e capaz de atender a múltiplas requisições simultâneas. Diversos SADs funcionam sobre NFSv3, porém os critérios de escolha precisam também considerar as características de funcionamento do FlexLab, como por exemplo, a necessidade de manter um compartilhamento montado mesmo após falhas na comunicação com o servidor.

Para o balanceamento de carga, foi adotado o sistema híbrido PVFS2 e pNFS. O sistema PVFS2, dentro de uma avaliação de SADs para sistemas de pequeno porte (BARBOSA, et al. 2007) incluindo o LUSTRE, OCFS2 com iSCSI e GlusterFS, é o que apresenta o melhor desempenho de escrita, embora esta não seja a predominância nas operações do *middleware* FlexLab.

Embora tenha bom desempenho em operações de leitura, o PVFS2 é inferior ao LUSTRE em ambientes de pequeno porte, como o que se apresenta para o FlexLab. Entretanto, como o PVFS2 não apresenta estratégia de *caching*, é também menos suscetível a falhas na conexão com o servidor. Mesmo em casos onde ocorrem perdas de conexão com o servidor, o sistema de arquivos montado sobre PVFS2 continua

funcionando, pois é capaz de restabelecer a conexão sem a necessidade de se reiniciar a MV, característica única dentre os SADs e de fundamental importância para o funcionamento do FlexLab.

O pNFS complementa o SAD criado pelo PVFS2, formando a camada responsável por trafegar todos os metadados da conexão da Camada 2, como informações sobre o servidor de origem e o nó de destino, próximo bloco requisitado e outras informações de confirmação de recebimento e agendamento do protocolo. O pNFS tem sua maior aplicação em redes WAN para acesso remoto a sistemas de arquivos paralelos, como o PVFS2. Entretanto, quando o pNFS é utilizado como no *middleware* do FlexLab, que atualmente funciona em rede LAN, oferece a Camada 2 uma conexão tolerante a falhas de rede conforme o tráfego aumenta em função do número de nós conectados (HILDEBRAND & HONEYMAN, 2007).

A Figura 21 mostra a arquitetura do Sistema de Arquivos Distribuídos adotado no *middleware* FlexLab. Quando a Camada 2 está configurada para utilizar o SAD baseado em pNFS/PVFS2, um novo servidor dedicado ao acesso da ISU é acrescentado a arquitetura.

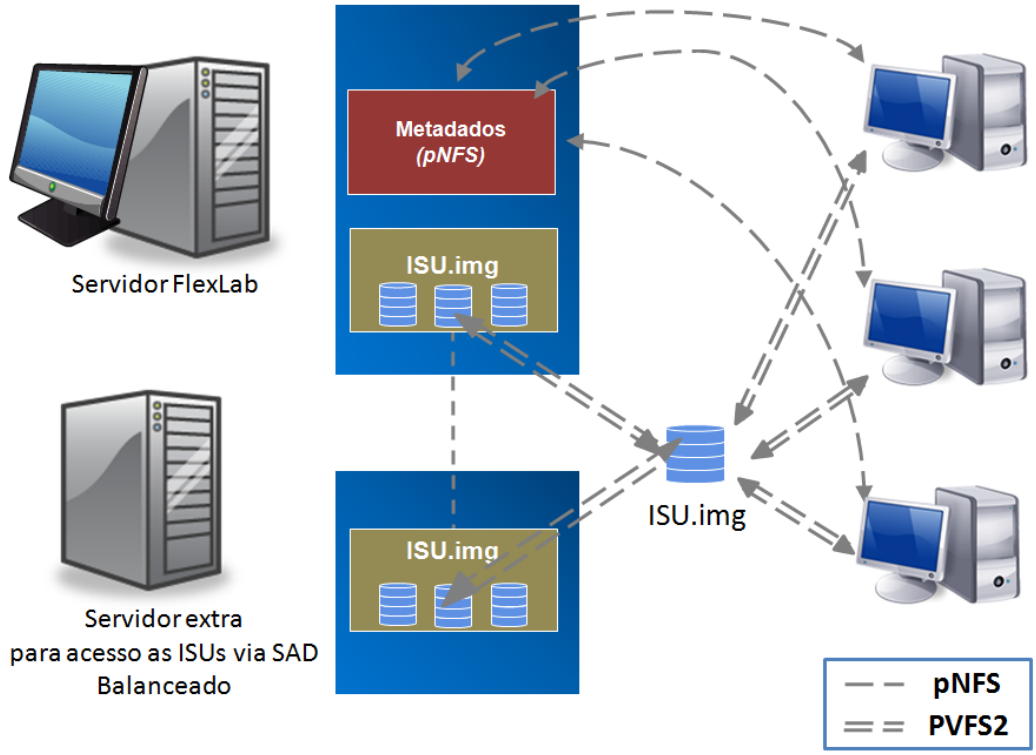

Figura 21: Arquitetura do Sistema de Arquivos Distribuídos com Balanceamento de carga do *middleware* FlexLab (AGUIAR, et al. 2008)(a).

5.7. Mecanismos de virtualização do *middleware* do FlexLab

Atualmente existem diversos tipos de MVs em nível de *hardware* que implementam os mecanismos de virtualização abordados no Capítulo 2. No Anexo A está disponível um quadro comparativo entre as MVs e sua possível utilização dentro do *middleware* FlexLab em função do atendimento às seguintes exigências:

- Mecanismo de virtualização que não exija uma versão modificada do SOC;
- Virtualização independente de recursos de virtualização específicos do processador;
- Disponibilidade da MV para execução sobre um SOH Linux;
- Permitir a execução de qualquer tipo de SOC;
- Permitir que a maior parte do código do usuário seja executada diretamente pelo processador;
- Permitir a emulação completa dos dispositivos de *hardware* de um computador (não simular dispositivos periféricos através de mapeamento da área de endereçamento real destes dispositivos);
- Possuir licença GPL, LGPL ou FreeBSD para fins de pesquisa;
- Permitir que a maior parte do código do *kernel* seja executado diretamente sobre o processador ou, quando não for possível, utilizar algum mecanismo de *trap* e emulação (não tratar código não-virtualizável através de extensões de virtualização do processador);
- Ter suporte a virtualização (disponibilidade de uma VMM) sobre a arquitetura x86.

A arquitetura do *middleware* do FlexLab permite que qualquer MV que atenda às exigências colocadas anteriormente sejam utilizadas na camada de virtualização. Embora seja uma possibilidade, desenvolver uma VMM por completo para atender as necessidades do *middleware* FlexLab é uma atividade extremamente onerosa e que não se justifica caso qualquer uma das MVs apresentadas como elegíveis ao *middleware* FlexLab do Anexo A apresentem um bom desempenho dentro do modelo de MVs Distribuídas proposto por este trabalho.

As MVs utilizadas na implementação do *middleware* FlexLab para este projeto foram o QEMU com o acelerador KQEMU (BELLARD, 2000) , o Xen (MENON, et al. 2006), o KVM (QUMRANET, 2006) e o VirtualBox (SUN MICROSYSTEMS, 2008). Embora todas estas MVs tenham sido avaliadas dentro do *middleware* FlexLab, determinadas características técnicas de implementação das VMMs inviabilizam o uso de algumas MVs de acordo com os requisitos estabelecidos anteriormente.

De todas as MVs consideradas, a única que se enquadra perfeitamente em todos os critérios estabelecidos pelo *middleware* FlexLab é a MV QEMU com o módulo de aceleração KQEMU. A Tabela 2 apresenta uma comparação entre as MVs Clássicas consideradas neste estudo.

Tabela 2: Comparativo[1] entre as MVs. Critério de seleção da MV para o FlexLab pode utilizar esta tabela como referência.

	Emulação	Tradução Binária	Assistência por Hardware	Paravirtualização
Compatibilidade com processadores (Exige extensões de virtualização?)	Excelente	Excelente	Fraca	Excelente
Compatibilidade de SO (Suporte a SO não modificado)	Excelente	Excelente	Excelente	Fraca
Desempenho	Fraco	Muito Bom	Excelente	Excelente
Sofisticação da VMM	Alta	Alta	Mediana	Mediana
Exemplos (MVs com licença LGPL e/ou GPL)	QEMU	QEMU+KQEMU e VirtualBox (PUEL)	KVM e XEN	XEN
Exemplos (MVs com licença proprietária)		VMware, VirtualBox e Virtual PC	VMware, XenApp e Virtual PC	XenApp

Das MVs consideradas para uso dentro do *middleware* do FlexLab, pode-se eliminar inicialmente todas que são disponibilizadas no modelo de licenciamento proprietário, pelo simples fato de não ser possível embarcar este tipo de MV dentro do *middleware* FlexLab sem ferir direitos de propriedade intelectual e industrial. Com o novo renascimento das MVs no mercado, ainda assim é possível encontrar uma boa gama de opções de MVs não proprietárias.

[1] Referência para o comparativo de desempenho: Jack Lo. **VMware and CPU Virtualization Technology**. VMWorld 2005, 2005. Disponível em www.wmware.com/vmi

Analisando as MVs não proprietárias, as que oferecem melhor desempenho, em função dos seus mecanismos de virtualização, são as VMMs Xen e KVM. A VMM da MV Xen permite ainda dois tipos de virtualização, a saber:

Paravirtualização: Exige o porte do sistema operacional convidado para que este possa executar uma versão modificada das chamadas de sistema a ABI, uma forma de permitir que o SOC execute diretamente sobre o processador real as instruções do modo *Kernel* que são de execução exclusiva do SOH, que executa em *Ring* 0 de privilégio.

***Full-virtualization* ou Virtualização Completa:** Neste modo, a VMM do Xen suporta a execução de sistemas operacionais não modificados, como o Microsoft Windows, entretanto exige que o processador tenha extensões de virtualização em sua arquitetura, como, por exemplo, as tecnologias VT-x (INTEL, 2006) e AMD-V (AMD, 2007). Em função das instruções adicionais e da arquitetura com suporte a virtualização destes processadores, o VMM pode executar chamadas de instruções executadas em *Ring* 0 (ou instruções privilegiadas do modo *Kernel*) que tradicionalmente não são virtualizáveis diretamente sobre o processador, melhorando muito o desempenho de execução.

Assim como o Xen, o sistema de virtualização adotado pela KVM também utiliza a virtualização completa, portanto, exige que o computador tenha um processador com as extensões de virtualização VT-x ou AMD-V. Embora apresentem o melhor desempenho de execução (com desempenho da MV próximo do desempenho nativo do processador), ambas as VMMs apresentam as seguintes deficiências para o *middleware* do FlexLab:

- Exigência do porte do SOC para suportar o modelo de paravirtualização (no caso do Xen);
- Exigência de um processador com suporte as instruções de extensão para virtualização (no caso do KVM e do Xen com VMM para Virtualização Completa);
- No caso do Xen, módulo do *Kernel* que dificulta o *boot* remoto utilizado na Camada 1 do FlexLab.

A Figura 22 mostra o mecanismo de funcionamento das extensões VT-x encontradas no processador Intel com suporte a virtualização. Com o suporte a virtualização em *hardware*, a VMM é capaz de usar a estrutura de controle VMX para agendar o acesso das MVs ao processador, permitindo desta forma que o SOCs sejam executados em *Ring* 0 de privilégio, da mesma forma que um SOH executaria em um sistema sem VT-x. Portanto, em VMMs executando sobre um processador sem suporte a virtualização, o SOC executa em *Ring* 3. Em um processador com o modo de operação VMX, a VMM executa em *Ring* 0.

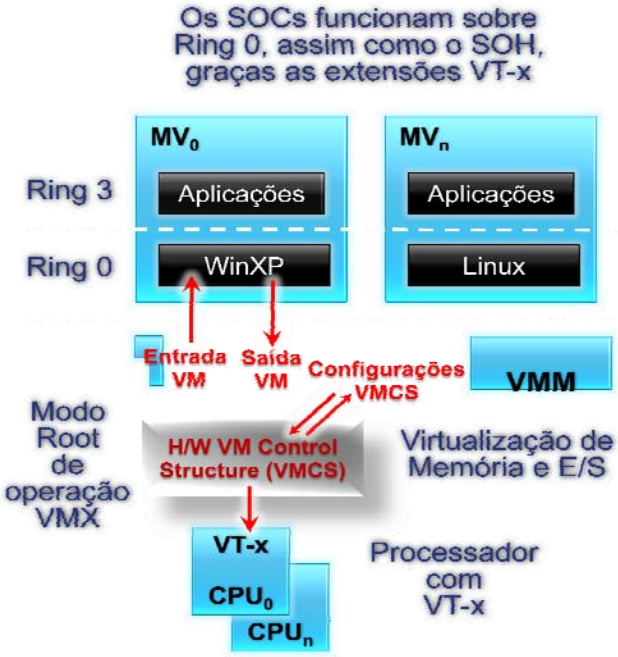

Figura 22: Mecanismo de deslocamento do *Ring* de execução do SOC (ROSEMBLUM & GARFINKEL, 2005).

A MV VirtualBox, ao contrário do Xen e do KVM, utiliza como mecanismo de virtualização a recompilação dinâmica de instruções, método baseado na emulação por tradução binária do QEMU. Assim como o KQEMU, a VirtuaBox executa a maior parte dos códigos de usuários e de *kernel* diretamente sobre o processador utilizando sua VMM. As semelhanças entre a VirtualBox e a QEMU também se estendem ao fato de que todo o modelo de emulação de dispositivos periféricos utilizado na VirtualBox também é baseado no modelo de dispositivos virtuais do QEMU. A principal diferença entre ambas está no formato como o VirtualBox executa suas MVs a partir do disco rígido virtual.

Para cada disco rígido virtual, a VMM do VirtualBox atribui um identificador universal único (UUID). Desta forma, não é possível atribuir um único disco rígido virtual (ou ISU) a diversas execuções concorrentes de MVs distribuídas pela rede, por exemplo. Para contornar este problema, é possível modificar a estrutura do descritor do arquivo que representa o disco rígido virtual do VirtualBox, para que ele seja marcado como imutável. Desta forma, mais de uma MV pode utilizar concorrentemente o mesmo arquivo de ISU e a partir deste momento, o disco virtual não poderá mais ser utilizado em modo de leitura e escrita, um recurso desejável no contexto do *middleware* FlexLab.

Soluções mais elaboradas podem ser desenvolvidas para permitir que um disco rígido virtual possa ser utilizado em modo *immutable* e *normal* ao se combinar duas cópias do arquivo com o disco rígido virtualizado, entretanto, antes de passar a discussão desta solução, vale apontar mais uma deficiência da VirtualBox.

Em sua versão de licenciamento baseado em código não proprietário, a VirtualBox não oferece suporte a virtualização da USB, um recursos essencial em um computador e, uma vez ausente da MV, não permite que o usuário utilize dispositivos baseados em USB. Esta limitação é contornada na versão proprietária da VirtualBox, que não pode ser utilizar no *middleware* do FlexLab.

Desta forma, por apresentar o mesmo nível de desempenho do QEMU + KQEMU, por em grande parte ter sido desenvolvida a partir deste projeto e por oferecer algumas limitações de uso no *middleware* do FlexLab, a MV escolhida para o projeto FlexLab e para os estudos deste trabalho foi a QEMU com o acelerador KQEMU.

Embora a KVM tenha sido inicialmente eliminada por funcionar apenas em computadores com processadores com extensões a virtualização, o fato do *middleware* do FlexLab poder alterar dinamicamente o tipo da MV utilizada nos permite propor que, para computadores dotados de processadores com as extensões VT-x e AMD-V suportadas pela KVM, o FlexLab possa alterar dinamicamente a MV para a KVM, de forma a permitir que o computador cliente execute o mais próximo possível do desempenho nativo do computador.

5.8. Funcionamento do FlexLab

As características da execução de um sistema operacional executado sobre o *middleware* do FlexLab são fiéis ao funcionamento de um sistema operacional

executado em um computador comum a partir do disco rígido local sem o mecanismo de virtualização. Durante os testes, foi utilizada uma ISU com a versão do sistema operacional Windows 2003 Server e aplicativos de escritório Microsoft Office. Todas as funcionalidades e recursos oferecidos pela ISU por meio do FlexLab mantiveram-se idênticas ao funcionamento de um computador normal, com exceção dos seguintes pontos:

- Pelo fato do *middleware* FlexLab ainda não tratar algumas teclas de atalho para serviços do QEMU, quando o usuário pressiona qualquer combinação válida das teclas "CTRL + ALT", por exemplo, "CTRL + ALT + F", ativa-se uma função da VMM, neste caso, a MV sai do modo tela-cheia e volta para o modo janela. No modo tela cheia, o usuário não visualiza nada além do próprio ambiente de tela do sistema operacional instalado. Com poucas alterações no código aberto da VMM é possível filtrar estas teclas para evitar a ativação acidental das funções da VMM por parte do usuário;
- As configurações de vídeo do SOC estão limitados aos recursos de vídeo emulados pela QEMU. Atualmente não é possível utilizar os recursos da placa aceleradora de vídeo, se houver, e a placa de vídeo emulada oferece resolução máxima de 1024 x 768 com 32 bits de cor;
- Ainda não foi desenvolvido um mecanismo de leitura de dispositivos USB suficientemente transparente como no computador normal. Atualmente é necessário montar o dispositivo USB manualmente no ambiente de execução do *middleware* (Fedora Core 3 modificado) para então exportar o dispositivo para o QEMU.
- Quando a IUS é iniciada em modo protegido, ou seja, não permite que o usuário altere de forma permanente o estado do disco rígido virtual, todos os dados salvos pelo usuário são perdidos no próximo *boot*.

Apesar destas limitações iniciais, também encontrada em todas as outras MVs, licenciadas ou de código fonte aberto, o funcionamento do FlexLab permite que qualquer tipo de aplicativo ou sistema operacional seja utilizado sem a necessidade de alteração em seu código fonte ou modificação na forma de instalação, tendo funcionamento, portanto, igual ao computador com execução nativa.

5.9. Considerações Finais

O *middleware* FlexLab é capaz de tirar proveito da combinação de três elementos essenciais para se oferecer uma solução de gerenciamento centralizado de computadores através de ISUs:

- Abstração de *hardware* oferecida pela camada de virtualização do seu *middleware*, permitindo que uma IUS funcione em qualquer conjunto de *hardware* compatível com o *middleware* FlexLab;
- Sistema de Arquivos Distribuído, que através de um sistema virtual paralelizado de arquivos e mecanismos de balanceamento de carga é capaz de oferecer diversos acessos concorrentes a um único arquivo contendo uma ISU;
- *Boot* remoto do *middleware* FlexLab com cópia direta para a memória RAM do computador cliente e *streaming* de *software* para acesso aos recursos do *middleware*, como o mecanismo dinâmico de inicialização de uma ISU.

6. EXPERIMENTOS

6.1. Considerações Iniciais

Esta Capítulo apresenta os resultados dos experimentos desenvolvidos neste trabalho e que consistiram em extrair uma base de informações de desempenho do tempo de inicialização do *middleware* FlexLab e da sua camada de virtualização, além de seu desempenho em função da técnica de virtualização adotada. Com os resultados obtidos dos experimentos foi possível compreender a viabilidade do modelo de virtualização distribuída proposto pelo *middleware* FlexLab e sua capacidade de escala.

6.2. Experimentos e Avaliação de Desempenho

O desempenho do *middleware* FlexLab é avaliado conforme dois parâmetros principais; desempenho do mecanismo de virtualização e suas opções e desempenho do Sistema de Arquivos Distribuídos e suas configurações. O objetivo dos experimentos é indicar a melhor configuração do sistema utilizando as tecnologias propostas para diminuir o tempo de *boot* da ISU e oferecer o melhor desempenho de execução após o *boot*.

Os experimentos utilizaram uma ISU de referência e também contemplaram a avaliação dos sistemas de arquivos distribuídos associado ao *middleware* FlexLab. Também foram consideradas extrações do tempo para iniciação de sistemas operacionais, tanto do SOH (*middleware*) quanto do SOC. Além disso, foram conduzidos em três tipos de ambientes, considerando as possibilidades de configuração do *middleware* e avaliando o tempo de *boot* do sistema e outros parâmetros de desempenho de execução de *benchmark* de teste, utilizando a suíte de *benchmark* SANDRA (SISOFTWARE, 2008).

6.3. Experimentos com o Sistema de Arquivos Distribuídos

As áreas críticas do *middleware* FlexLab que fazem acesso constante ao servidor foram configuradas para terem seus dados alocados diretamente na RAM do cliente, dessa forma, as operações mais executadas no *middleware* já se encontram na memória e não precisam ser requisitadas ao servidor novamente. Experimentos realizados em 10 computadores na configuração da Tabela 3 com apenas um servidor demonstraram que o fluxo de dados na rede de comunicação diminui em 37% para pacotes UDP após as alterações de *cache* no *middleware*.

Tabela 3: Configuração do ambiente de avaliação do sistema de *boot* do *middleware* FlexLab

Tipo	Configuração
Servidor 1	**Processador Dual Core E4500 2.2GHz, 1 GB de RAM, Disco rígido de 120 GB SATA 7200 RPM, rede de 1GB/s**
Servidor 2	**Processador Dual Core E4500 2.2GHz, 1 GB de RAM, Disco rígido de 120 GB SATA 7200 RPM, rede de 1GB/s**
Clientes	**Processador Dual Core E4500 2.2GHz, 1 GB de RAM, Disco rígido de 120 GB SATA 7200 RPM, rede de 1GB/s**
Switch	**100Mb/s**

O tamanho do bloco UDP foi configurado para 8 KB ao invés de 4 KB enviando mais dados por pacote como visto em Kuacharoen (KUACHAROEN, 2002). O *software* utilizado para verificar o fluxo de rede foi o ntop (NTOP, 2007). Após as alterações no *middleware* o tempo de *boot* diminuiu aproximadamente 10 segundos em relação ao *boot* original, como visto na Figura 15.

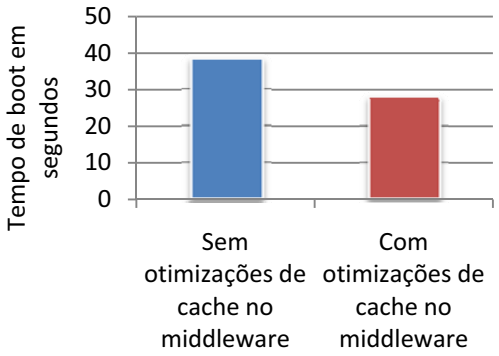

Figura 23: Tempo de *boot* para 10 clientes utilizando otimizações na *cache* do *middleware* FlexLab (AGUIAR, et al. 2008)(b)

6.4. *Boot* do FlexLab

Os experimentos de *boot* se basearam na extração de informações de tempo na inicialização das máquinas remotas. Os valores são capturados através de *scripts* que executam a leitura do *log* do DHCP e armazenam o tempo total de inicialização. Para garantir que os resultados sejam precisos é necessário que todos os computadores estejam com os horários do sistema operacional sincronizados, sendo utilizado para isso o *Network Time Protocol* (NTP) (NTP, 1998).

Para otimizar o desempenho do acesso aos arquivos de máquinas virtuais na Camada 2 do *middleware* FlexLab, foram utilizadas configurações de *cache* no *middleware* (ZHAO, et al. 2006) e nas implementações do PVFS2/pNFS favorecendo o acesso aos arquivos contínuos utilizados como ISUs.

O tamanho do bloco de transmissão foi configurado para 8 KB no *boot* do *middleware* (UDP) e 36 KB para o *boot* das máquinas virtuais (TCP) (KUACHAROEN, 2002), permitindo que a inicialização da ISU fosse mais estável (ou seja, com menos falhas durante o *boot* dos computadores). Imagens contínuas, como em máquinas virtuais (ZHAO, et al. 2006), possuem maior velocidade de acesso paralelizado quando o SAD utiliza blocos maiores de *cache*. Aumentando o tamanho do bloco de 8KB no *middleware* para 36 Kbytes para a máquina virtual favorece a velocidade de *boot* FlexLab em função das características de funcionamento do SAD.

A Tabela 4 mostra o tempo de *boot* de um ambiente virtualizado e distribuído em que os computadores clientes são inicializados simultaneamente por *Wake-On-Lan*, sendo o Windows 2003 o sistema operacional virtualizado utilizado na ISU. O sistema de arquivos distribuído na Camada 1 do FlexLab é o NFSv3 e na Camada 2 o PVFS2/pNFS.

O tempo de *boot* é a soma do tempo do *middleware* (*kernel* Linux + MV) mais o tempo real de iniciação da ISU. Com o experimento é possível notar a queda de 34 segundos no tempo de *boot* quando se aumenta o número de clientes de um para trinta computadores utilizando-se apenas dois servidores para balanceamento de carga na camada de SAD do FlexLab.

Tabela 4- Tempo de *boot* em Segundos de uma ISU composta por um Windows 2003 virtualizado e distribuído para um 1 a 30 computadores clientes.

Cenários	Middleware FlexLab	Inicialização da Máquina Virtual	Tempo Total de *Boot*
1 cliente	18	39	57
5 clientes	23	55	78
10 clientes	28	69	98
30 clientes	108	245	353

Para validar o experimento foram criados três diferentes modelos de *boot*, como visto na Tabela 5, com tempo de *boot* tradicional a partir de uma instalação local, tempo de *boot* de um sistema virtualizado a partir do disco rígido local e tempo de *boot* virtualizado a partir do acesso remoto. Usando como comparativo um ambiente com um computador, foi possível constatar que o modelo distribuído e virtualizado utilizado no FlexLab é 24 segundos mais lento que um sistema operacional inicializado do disco rígido de forma tradicional e 18 segundos mais lento que o sistema operacional virtualizado a partir do disco rígido. Entretanto, este valor inclui o tempo de *boot* do *middleware* (Camada 1), que consome por volta de 22 segundos do tempo total de *boot* da ISU.

Tabela 5 - Comparativo de tempo de *boot* em Segundos com um computador cliente (AGUIAR, et al. 2008)(a)

Modelo	Acesso a Imagem	Tempo (segundos)
Windows 2003 Sem Virtualização	Disco Rígido	40
Windows 2003 Virtualizado	Disco Rígido	46
Windows 2003 Virtualizado e Distribuído	Remoto	64

6.5. Desempenho da Camada de Virtualização

A camada de virtualização impõe um preço pelo benefício da abstração de *hardware* na forma de um decréscimo perceptível no desempenho do computador. Conforme apresentado no Capítulo 5, para atender os requisitos de funcionamento do *middleware* FlexLab, foram considerados para fins de testes os mecanismos de virtualização baseados em QEMU e o acelerador KQEMU, que utilizam tradução binária dinâmica e apresentam o maior nível de compatibilidade de *software* e

hardware. O objetivo destes experimentos é avaliar o impacto que a camada de virtualização causa no desempenho do *middleware* FlexLab e compara os mecanismos de virtualização. O ambiente de teste utilizado nos testes da camada de virtualização é apresentado na Tabela 6:

Tabela 6: Configuração dos computadores do ambiente de testes para a camada de virtualização

Tipo	Configuração
Servidor 1	**Processador Dual Core E4500 2.2GHz, 1 GB de RAM, Disco rígido de 120 GB SATA 7200 RPM, rede de 1GB/s**
Servidor 2	**Processador Dual Core E4500 2.2GHz, 1 GB de RAM, Disco rígido de 120 GB SATA 7200 RPM, rede de 1GB/s**
Clientes	**Processador Dual Core E4500 2.2GHz, 1 GB de RAM, Disco rígido de 120 GB SATA 7200 RPM, rede de 1GB/s**
Switch	**100Mb/s**

A partir do ambiente criado para os experimentos em 6.3, portanto com o SAD baseado em pNFS/PVFS2 com balanceamento de carga, foi possível observar o desempenho do *middleware* do FlexLab baseado na camada de virtualização QEMU com as opções:

- Aceleração KQEMU em modo *kernel* e usuário;
- Aceleração KQEMU somente usuário;
- Sem aceleração.

A partir destes experimentos é possível avaliar o impacto do mecanismo de virtualização no desempenho do SAD e do *middleware*. Para um número inferior a cinco computadores, é possível observar uma diferença inferior a 3 segundos no tempo total de *boot* do sistema. Entretanto, com o maior número de computadores iniciando suas ISUs concorrentemente, o melhor desempenho de virtualização oferecido pelo mecanismo baseado no acelerador KQEMU permite que o desempenho global do sistema seja superior em relação ao *boot* com um sistema de virtualização não baseada em VMM clássica, conforme mostra a Figura 24. Desta forma, conclui-se que o melhor desempenho da camada de virtualização contribui com o desempenho global de *boot* em um ambiente.

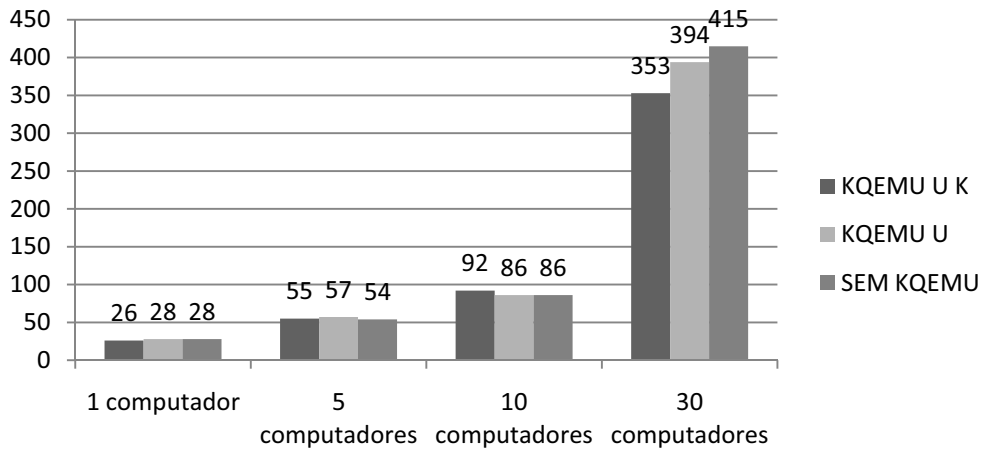

Figura 24: Comparação do tempo de *boot* em segundos em função do mecanismo de virtualização

Considerando ainda as configurações de *hardware* da Tabela 6, o estudo também avaliou a degradação de desempenho causada pela camada de virtualização utilizada no FlexLab. Para isso, foi utilizada como forma de medida o tempo de abertura do aplicativo Microsoft Word 2003, medido através da aplicação de medição de tempo Intel VTune Performance Analyzer 9.0. Os seguintes cenários foram avaliados: Execução nativa do aplicativo Microsoft Word 2003 e execução do mesmo aplicativo nos computadores que realizaram o *boot* remoto por meio do FlexLab. No caso do *boot* remoto através do FlexLab, foi utilizado na camada de virtualização a MV QEMU com o acelerador KQEMU.

Tabela 7: Tempo (em segundos) de abertura do aplicativo Microsoft Word 2003 para observação da degradação de desempenho causada pela camada de virtualização do FlexLab

Cenários	Execução Nativa (*Boot* local)	QEMU+KQEMU (*Boot* pelo FlexLab)	Degradação de Desempenho
1 cliente	4,2	4,4	0,2
5 clientes	4,2	4,5	0,3
10 clientes	4,2	4,5	0,3
30 clientes	4,2	4,7	0,5

Observa-se na Tabela 7 que o a degradação de desempenho causada pela camada de virtualização do FlexLab é inferior a 5% considerando a execução de apenas 1 cliente. Mesmo no cenário com 30 clientes executando o aplicativo ao mesmo tempo por meio do Sistema de Arquivos Distribuído, a degradação de desempenho é inferior a 1 segundo, representando uma degradação de 11% no tempo de execução do aplicativo.

6.6. Desempenho do FlexLab em ambientes heterogêneos

Em função da existência da camada de virtualização, um dos recursos do FlexLab é o *boot* de uma ISU em um ambiente composto por diferentes conjuntos de *hardware*. Desta forma, o gerenciamento do FlexLab pode ser centralizado a uma única imagem administrativa que pode ser distribuída para execução em diferentes tipos de computadores. Os testes foram desenvolvidos a partir de um ambiente heterogêneo composto por 30 computadores, dos quais sete com configuração diferente de *hardware* conforme indica a Tabela 7.

Tabela 8: Tipos de configurações dos computadores do ambiente heterogêneo de teste

Tipo	Quantidade	Configuração
Servidor 1	1	**Processador Dual Core E4500 2.2GHz, 1 GB de RAM, Disco rígido de 120 GB SATA 7200 RPM, rede de 1GB/s**
Servidor 2	1	**Processador Dual Core E4500 2.2GHz, 1 GB de RAM, Disco rígido de 120 GB SATA 7200 RPM, rede de 1GB/s**
Cliente 1	24	**Processador Dual Core E4500 2.2GHz, 1 GB de RAM, Disco rígido de 120 GB SATA 7200 RPM, rede de 1GB/s**
Cliente 2	2	**Processador Pentium M 2.0GHz, 1 GB de RAM, Disco rígido de 120 GB SATA 5400 RPM, rede de 1GB/s**
Cliente 3	3	**Processador Pentium 4 1.8 GHz, 512 MB de RAM, Disco rígido de 40 GB PATA 4200 RPM, rede de 100 MB/s**
Cliente 4	1	**Processador Pentium M 1.73 GHz, 1 GB de RAM, Disco rígido de 40 GB SATA 5400 RPM, rede de 1GB/s**
Switch	1	**100Mb/s**

Os testes desenvolvidos comparando o ambiente homogêneo com o ambiente heterogêneo de computadores utilizaram a MV QEMU com o acelerador KQEMU em modo de execução *kernel* e usuário associado ao Sistema de Arquivos Distribuído na Camada 2 e mostraram que o ambiente heterogêneo com computadores com menor poder de processamento degradaram o desempenho em 9%.

Tabela 9: Comparação do tempo de *boot* em segundos de um ambiente heterogêneo contra um ambiente homogêneo (30 computadores em cada caso).

Ambiente com 30 computadores (tempo em segundos)	Tempo de Middleware	Tempo da MV	Tempo Total de *Boot*
Ambiente Homogêneo	132	225	357
Ambiente Heterogêneo	128	262	390

6.7. Considerações Finais

O FlexLab pode ser aplicado em diversos cenários, não se limitando apenas ao gerenciamento de computadores. Entre os possíveis cenários de aplicação estão:

- Gerenciamento de redes de computadores de uso corporativo onde os requisitos de padronização de configuração de *software* sejam elevados e os usuários demandem aplicativos pouco convenientes para arquiteturas *thin-clients* (por exemplo, *software* para edição de vídeos ou CAD);
- Gerenciamento de computadores de uso compartilhado, como laboratórios de informática, onde a necessidade de padronização de configuração de *software* simplifica as tarefas administrativas e reduz o custo de manutenção (MORGADO, CRUZ & TWANI, 2008). O fato dos computadores poderem também fazer o *boot* de diferentes tipos de imagens e versões de ISUs permite que o laboratório de uso compartilhado seja utilizado por diversas turmas com diferentes configurações de imagens;
- Gerenciamento de LAN *Grids* compostas por diversas MVs distribuídas pelo FlexLab para o processamento de atividades em segundo plano utilizando a capacidade ociosa de processamento dos computadores dos usuários;
- Criação de *Clusters* multiuso a partir de computadores *commodity*, permitindo que computadores com capacidade ociosa de processamento (por exemplo, não são utilizados no período da noite) sejam utilizados para fins de computação de alto-desempenho. Neste caso, o FlexLab pode distribuir as MVs com o ambiente de *Cluster* já preparado e configurado.

De acordo com as atividades de experimentação do *middleware* FlexLab desenvolvidas por este projeto, é possível afirmar que a implantação de um SAD otimizou o desempenho em operações de leitura e permitiu maior potencial de escala da solução, apresentando ainda a possibilidade de execução de uma ISU em ambientes heterogêneos de computadores. O desempenho da camada de virtualização pode influenciar de forma significativa o tempo de execução e de *boot* da ISU, e mostra que um melhor mecanismo de virtualização pode favorecer o desempenho global do ambiente de MVs distribuídas criado pelo FlexLab.

7. CONCLUSÃO

A combinação das técnicas de virtualização de *hardware* com um Sistema de Arquivos Distribuídos permite o desenvolvimento de uma solução de gerenciamento de ambientes computacionais capaz de oferecer a administração das configurações de *software* através de uma ISU, sem com isso sacrificar o desempenho de execução de cada cliente em função do número de computadores conectados ao sistema, situação oposta ao que ocorre em soluções do tipo *thin-client*. Os desafios, entretanto, são oferecer todos estes benefícios conjugados em um *middleware* transparente, otimizado e que ofereça suporte a escala e desempenho para uma arquitetura de baixo custo, como é o caso da arquitetura x86. Técnicas de virtualização recentes, como os modelos de virtualização assistida por *hardware*, indicam que é possível oferecer uma solução baseada em virtualização distribuída com um decréscimo de desempenho compatível com o desempenho da execução nativa, o que significa oferecer aos computadores clientes a possibilidade de execução de uma ISU virtualizada com desempenho próximo ao nativo do processador.

Ao avançar sobre o paradigma de virtualização distribuída utilizando um modelo multicamada, o FlexLab oferece os benefícios apresentados pela virtualização e aproveita a evolução constante dos mecanismos de virtualização, como as VMMs com suporte a virtualização assistida por *hardware*, uma vez que o modelo multicamada permite que o *middleware* de virtualização seja alterado em tempo de *boot*. Além disso, o modelo FlexLab traz os benefícios da flexibilidade de execução sobre plataformas heterogêneas com operação distribuída através de discos virtuais.

O estudo mostra que uma solução como o FlexLab, baseada em virtualização distribuída, é capaz de oferece um *middleware* viável para gerenciamento de computadores x86 por meio de uma ISU em função dos resultados apresentados.

Embora o *middleware* do FlexLab introduza uma degradação no desempenho de execução da ISU, esta degradação é linear em função do número de nós adicionados ao servidor do FlexLab.O crescimento é linear uma vez que o tempo de execução de aplicativos está em sua maior parte associado aos recursos locais de processamento do computador, e não aos recursos do servidor do FlexLab. Desta forma, a degradação de desempenho para a execução de aplicativos em um computador utilizando o FlexLab

como ambiente de execução é próxima a degradação de desempenho encontrada em uma rede com 30 computadores utilizando o ambiente de execução do FlexLab. Nos testes desenvolvidos pelo projeto, a degradação de desempenho não foi superior a 11% do tempo de execução nativa para um computador.

Ao apresentar os benefícios do gerenciamento através de uma ISU mesmo em ambientes com configuração heterogênea de *hardware* associado a uma degradação de desempenho aproximadamente constante, o FlexLab mostra-se uma alternativa viável como ferramenta de gerenciamento centralizado. Com a evolução no poder de processamento dos novos processadores, a redução do custo destes componentes e a disponibilização de mecanismos de virtualização assistidos por *hardware*, esta degradação de desempenho tende a ser ainda mais baixa, criando a possibilidade para que o modelo de gerenciamento através de virtualização distribuída do FlexLab possa oferecer todos os benefícios apresentados neste projeto, porém com um desempenho similar a execução nativa do processador.

7.1. Contribuições deste Trabalho

Os estudos para o desenvolvimento do FlexLab conduziram a criação de um mecanismo de gerenciamento de computadores utilizando a abordagem da virtualização distribuída por meio de uma arquitetura multicamada, contribuindo para pesquisas acadêmicas futuras:

- Um estudo bibliográfico sobre as principais tecnologias de virtualização, sistemas distribuídas e gerenciamento por meio de *boot* remoto.

- Proposta de uma arquitetura multicamada baseada em um sistema de arquivos distribuído com balanceamento de carga, viabilizando a utilização de ISUs no modelo de *boot* remoto.

- Uma análise do uso em aplicações de *cluster* multiuso aproveitando recursos ociosos em laboratórios de informática e estações de trabalho.

- A utilização de um *middleware* de virtualização distribuída para o gerenciamento de computadores por meio de ISUs, explorando um paradigma

descentralizado de gerenciamento por meio de virtualização diferentemente dos mecanismos adotados pelas soluções atuais. .

7.2. Trabalhos Futuros

A partir do *middleware* FlexLab de virtualização distribuída com aplicação em gerenciamento de computadores corporativos e criação de *Clusters* multiuso, oportunidades de trabalhos futuros são citados:

- Adaptação do *middleware* FlexLab para funcionamento através da Internet, permitindo assim que seus usuários possam utilizar qualquer computador como cliente do FlexLab ao redor do mundo;
- Adaptação do *middleware* FlexLab para funcionamento em redes locais sem-fio;
- Aprimoramentos no mecanismo de virtualização para que a utilização de dispositivos pela porta USB seja automatizada e transparente para o usuário;
- Utilização do possível disco rígido físico local existente no computador do cliente como mecanismo de *cache* de arquivos, aplicando o algoritmo rsync para otimizar o tráfego de dados na rede;
- Aprimorar o recurso de balanceamento de carga para que o número de novos clientes adicionados na rede do FlexLab não dependa do número de servidores de ISU. Desta forma, outros nós da rede FlexLab podem se tornar servidores de ISU;
- Desenvolver uma topologia P2P para o Sistema de Arquivos Distribuídos do FlexLab para aplicação do *middleware* em redes virtuais sobre a Internet;
- Explorar as possibilidades de uso da EFI como sistema para hospedar os serviços do *middleware* FlexLab, oferecendo recursos de virtualização nativos ao computador;

8. REFERÊNCIAS

ABADALA, S.; CHADHA, V.; CHAWLA, P.; FIGUEIREDO, R.; FORTES, J.; KRSUL, I.; MATSUNAGA, A.; TSUGAWA, M.; ZHANG, J.; ZHAO, M.; ZHU, L.; ZHU, X. (2005). **From Virtualized Resources to Virtual Computing *Grids*: the In-VIGO System.** *Future Generation Computer Systems.* 2003, Vol. 21, Issue 6, P. 896-909.

ADAMS, K.; AGESEN, O. (2006). **A Comparison of *Software* and *Hardware* Techniques for x86 Virtualization.** *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems ASPLOS'06*. 2006, San Jose, Califórnia, p. 2-13.

AGUIAR, C.S. (2008). **Modelo de Virtualização Distribuída Aplicado ao Gerenciamento e Replicação de Clusters Multiuso.** *Universidade Estadual Paulista "Júlio de Mesquisa Filho", Pós-Graduação em Ciências da Computação*. 2008.

AGUIAR, C. S.; CRUZ, D. I.; ULSON, R. S.; CAVENAGHI, M. A. (2008). **The application of Distributed Virtual Machines for Enterprise Computer Management: a two-tier network file system for image provisioning and management.** *32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC 2008).* 2008, p. 428 -431. (a).

AGUIAR, C. S.; CRUZ, D. I.; ULSON, R. S.; CAVENAGHI, M. A. (2008). **The application of Virtualized Multiuse Clusters for LAN *Grid* Network Computer Management.** *Poster in 8th IEEE International Symposium on Cluster Computing and Grid (CCGRID 2008)*, 2008 (b).

ALTMAN, E.; GSCHWIND, M.; SATHAYE, S.; KOSONOCKY, S.; BRIGHT, A.; FRITTS, J. (1999). **BOA: The Architecture of a Binary Translation Processor.** *Research Report RC21665, IBM T.J. Watson Research.* 1999.

ANDERSON, D.; CHASE, J.; (2000). **Failure-Atomic File Access in an Interposed Network Storage System**. *Ninth IEEE International Symposium on High Performance Distributed Computing HPDC 2000.* 2000. p. 157.

ARDENCE. (2007). **Ardence *Software* Streaming Platform: *Desktop* Edition**. Disponível em: <http://www.ardence.com>. Acesso em: Outubro 2007.

BARBOSA, A. A,; GREVE, F.; BARRETO, L.P. (2007). **Avaliação de Sistemas de Arquivos Distribuídos num Ambiente de Pequena Escala.** *Anais do XXVII Congresso da SBC, Rio de Janeiro, RJ*. Junho de 2007, p. 852.

BARHAM, P.; DRAGOVIC, B.; KEIR, F. (2003). **Xen and the Art of Virtualization.** *ACM Symposium on Operating Systems Principles.* 2003, New York, USA. p. 164 – 177.

BARATTO, A. R.; KIM, N. L.; NIEH, J. (2005). **THINC: A Virtual Display Architecture for *Thin-client* Computing.** *ACM Symposium on Operating Systems Principles. 2005. Brighton, United Kingdom*. 2005, p. 277 - 290.

BELLARD, F. (2005). **QEMU, a Fast and Portable Dynamic Translator.** *2005 USENIX Annual Technical Conference. USENIX Association.* 2005. p.14 - 21.

BREY, B. (2003). ***The Intel Microprocessors. Architecture, Programming and Interfacing.*** Prentice Hall, 6th Edition. 2003.

CHERVENAK, A.; FOSTER, I.; KESSELMAN, C.; SALISBURRY, C.; TUECKE, S. (2000). **The data *Grid*: Towards an architecture for the distributed management and analisys of large scientific datasets**. *Journal of Network and Computer Applications (2000).* Academic Press, Vol. 23, Issue 3, p. 187 -200.

CITRIX. (2008)**.** Disponível em: http://www.citrix.com.br/products/provisioning_server_*desktop*s.php. Acesso em 13 de Março de 2008.

DODONOV, E. (2006). **Modelo para a Avaliação e Predição de Comportamento de Processos aplicado a Sistemas de Memória Compartilhada Distribuída.** *Dissertação de Mestrado em Ciência da Computação - ICMC USP – São Carlos*. Agosto 2006.

ELENBOGEN, S. B. (1999). **Computer Network Management: theory and practice**. *ACM SIGCSE Bulletin.* Vol. 31, nº 1, March 1999, p. 119 – 121.

FARRELL, C.; KIERONSKA, D.; KORDA, D. (1995). **Performance Implications of Virtualisation of Massively Parallel Algorithm Implementation.** *IEEE First International Conference on Algorithms and Architectures for Parallel Processing.* 1995, p. 842.

FIGUEIREDO, R.; DINDA, P.; FORTES, J. (2003). **A Case for Grid Computing On Virtual Machines**. *ICDCS, Proceedings of the 23rd International Conference on Distributed Computing Systems, IEEE*. 2003, p. 550.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. (2001) **The Anatomy of the Grid: Enabling Scalable Virtual Organization.** *International Journal of High Performance Computing Applications.* 2001, Vol. 15, No. 3, p. 200 – 220.

GOLDBERG. R. (1973). **Architecture Of Virtual Machines.** *Honeywell Information Systems, Inc. Billerica, Massachusetts and Harvard University Cambridge, Massachusetts. Proceedings of Workshop on Virtual Computer Systems, Cambridge, MA*. 1973, p. 74-112.

HENRY, M. (2000). **PXE Manageability Technology for EFI**. *Intel Developer Update Magazine, Intel,* Outubro de 2000, p. 3. Disponível em http://www.intel.org/technology/magazine/systems/it10004.pdf. Acesso em 14 de Novembro de 2007.

HILDEBRAND, D.; HONEYMAN, P. (2007). **Direct-pNFS: scalable, transparent, and versatile access to parallel file systems.** *Proceedings of the 16th International Symposium on High Performance Distributed Computing, Monterey, CA, USA.* 2007, p. 199 – 208.

INTEL. (2007). **Pre*boot* Execution Environmet (PXE) Specification Version 2.1**.Disponível em http://www.pix.net/*software*/pxe*boot*/archive/pxespec.pdf. Último acesso em 27 de Novembro de 2007.

KANEDA, K.; OYAMA, Y.; YONEZAWA, A. (2005). **A Virtual Machine Monitor for Utilizing Nondedicated Clusters.** *University of Tokyo, 731 Hongo Bunkyoku Tokyo, Japan. SOSP'05.* 2005. p. 533.

KING, S.; CHEN, P. **Operating System Extensions to Support Host Based Virtual Machines.** (2002). *Department of Electrical Engineering and Computer Science University of Michigan.* Disponível em: http://www.eecs.umich.edu/techreports/cse/2002/CSE-TR-465-02.pdf. 2002. Último acesso em Outubro 2007.

KRSUL, I.; GANGULY, A.; ZHANG, J.; FORTES, J.; FIGUEIREDO, R. (2004). **VMPlants: Providing and Managing Virtual Machine Execution Environments for *Grid* Computing.** *Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference.*2004. p. 7 – 21.

KUACHAROEN, P.(2004). **Embedded *Software* Streaming via Block Streaming**, *Dissertação apresentada para o Programa de Pós-graduação do Georgia Institute of Technology, School of Electrical and Computer Engineering.* Abril 2004.

LEVASSEUR, J.; UHLIG, V.; CHAPMAN, M.; CHUBB, P.; LESLIE, B.; HEISER, B. (2006). **Pre-virtualization: soft layering for virtual machines. *U**niversity of Karlsruhe, Germany, IBM T. J. Watson Research Center, NY*.** 2006.

MENON, A.; COX, A.; ZWAENEPOEL, W. **Optimizing Network Virtualization in Xen.** *Proceedings of 2006 USENIX Annual Technical Conference*. 2006. p. 15 – 28.

MENON, A.; SANTOS, J.; TURNER, Y.; JANAKIRAMAN, G.; ZWAENEPOEL, W. (2005). **Diagnosing Performance Overheads in the Xen Virtual Machine Environment.** *First ACM/USENIX Conference on Virtual Execution Environments VEE'05,* Junho 2005. p. 13.

MENON, A.; SANTOS, J.; TURNER, Y.; JANAKIRAMAN, G.; ZWAENEPOEL, W. (2003). **Xen and the Art of Virtualization.** *19th ACM Symposium on Operating Systems Principles SOSP'03*, October 19–22, 2003, Bolton Landing, New York, USA. p. 164.

MICROSOFT. (2007). **Microsoft Windows XP**. Disponível em http://www.microsoft.com/brasil/windowsxp/default.mspx. Acesso em 14 de Agosto de 2007.

MORGADO, E.; CRUZ, D. I.; TWANI, E.; (2008). **Paving the Way for a Dynamic and Mature ICT Infrastructure in Education: A Case for Schools in Emerging Markets.** *Proceedings of the International Conference on Engineering and Technology Education - INTERTECH 2008.* 2008, p. 79 - 87.

ntop. (2007). ntop. Disponível em http://www.ntop.org/. Acesso em 13 de Março de 2008.

POPEK, G.; GOLDBERG, R. (1974). **Formal Requirements for Virtualizable Third Generation Architectures.** *Communications of the ACM, Vol 17, Issue 7.* 1974. p. 412 - 421.

QUMRANET. (2008). **KVM – *Kernel*-based Virtualization Machine.** White Paper, 2006. Disponível em http://www.qumranet.com/art_images/files/8/KVM_Whitepaper.pdf. Acessado em 17 de Fevereiro de 2008.

ROSENBLUM, MENDEL. (2004). **The Reincarnation of Virtual Machines.** *ACM Queue vol. 2, n. 5.* 2004. p. 17.

ROSENBLUM, M.; GARFINKEL, T. (2005).**Virtual Machine Monitors: Current Technology and Future Trends,** *IEEE Computer. Vol. 38, Issue 5.* May 2005. p. 39.

RUTH, P.; JIANG, X.; XU, D.; GOASGUEN, S. (2005). **Virtual Distributed Environments in a Shared Infrastructure.** *Purdue University. IEEE Computer Society*. May 2005. p. 63.

SAPUNTZAKIS, C.; CHANDRA, R.; PFAFF, B.; LAIN, M.; ROSENBLUM, M.; CHOW, J. (2003). **Optimizing the Migration of Virtual Computers.** *Computer Science Department Stanford University. 5th Symposium on Operating Systems Design and Implementation.* 2003. p. 377.

SILBERSCHATZ, A.; PETERSON, J. L.; GALVIN, P. B. (1991). ***Operating system concepts***. 3.ed. Reading, USA: Addison-Wesley, 1991. p. 696.

SIRER, E.; GRIMM, R.; BERSHAD, B.; GREGORY, A.; MCDIRMID, S. (1998). **Distributed Virtual Machines: A System Architecture for Network Computing**. *Proceedings of the 8th ACM SIGOPS European workshop on Support for composing distributed applications*. 1998. p. 13-16.

SI*SOFTWARE*. (2008)**. SANDRA *Software* Benchmark**. Disponível em http://www.si*software*.net/. Acesso em 14 de Março de 2008.

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

SUGERMAN, J.; VENKITACHALAM, G.; LIM, B. (2001). **Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor.** *Proceedings of the 2001 USENIX Annual Technical Conference, Boston, MA*. 2001.p. 1 – 14.

SUN MICROSYSTEMS. (2008). **VirtualBox.** Disponível em: http://www.virtualbox.org/wiki/GPL. Acesso em 16 de Fevereiro de 2008.

TRIDGELL, A.; MACKERRAS, P. (1996). **The rsync algorithm.** *The Australian National University, Canberra, ACT 0200, Australia.* June 1996. Disponível em http://cs.anu.edu.au/techreports/1996/TR-CS-96-05.pdf. Último acesso em Outubro de 2007.

UHLIG, R.; NEIGER, G.; RODGERS, D.; SANTONI, A.; MARTINS, F.; ANDERSON, A.; BENNETT, S.; KÄGI, A.; LEUNG, F.; SMITH, L. (2005). **Intel Virtualization Technology.** *IEEE Computer Society Vol. 38, Issue 5.* 2005. p. 48-56.

VMWARE. (2008). Disponível em: **<**http://www.vmware.com**>**. Acesso em: 14 de Abril de 2008.

VMWARE. (2007). **Virtual *Desktop* Infra-structure.** Disponível em: http://www.vmware.com/pdf/virtual_*desktop*_infrastructure_wp.pdf. Acesso em Novembro de 207.

WALDSPURGER, C. (2003). **Memory Resource Management in VMware ESX Server.** *5th Symposium on Operating Systems Design and Implementation*. 2003. p. 181.

WHITAKER, A.; COX, R.; SHAW, M.; GRIBBLE, S. (2005). **Rethinking the Design of Virtual Machine Monitors.** *University of Washington. IEEE Computer Society. Vol. 38, Issue 5.* May 2005. p. 57 - 62.

WHITAKER, A.; SHAW, M.; GRIBBLE , D. S. (2002) **Scale and performance in the Denali isolation *kernel.*** *ACM SIGOPS Operating Systems Review, Vol. 36, Issue SI,* December 2002, p. 195 – 209.

YANG, C. Y.; CHEN, P.; CHEN, Y. (2005). **Performance Evaluation of SLIM and DRBL Diskless PC Clusters on Fedora Core 3**. *Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies. Washington, DC, USA.* 2005, p. 479 – 482.

YAP, K.; BAUM, G. (2003). ***EtherBoot developers' manual.*** Disponível em: http://www.ether*boot*.org/wiki/dev/devmanual. Último acesso em Outubro 2007.

ZHAO, M.; ZHANG, J.; FIGUEIREDO, R.; (2006). **Distributed File System Virtualization Techniques Supporting On-Demand Virtual Machine Environments for *Grid* Computing.** *Advanced Computing and Information Systems Laboratory, Electrical and Computer Engineering, University of Florida, Gainesville, Florida 32611, USA, Springer Science + Business Media, Inc. Cluster Computing 9*, 2006. p. 45–56.

ZHOU, Y.; ZHANG, Y.; XIE, Y. (2006). **Virtual Disk based Centralized Management for Enterprise Networks.** *SIGCOMM'06 Workshops, Pisa, Italy. ACM.* 2006. p. 23.

ANEXO A

Tabela 1: Quadro comparativo das Máquinas Virtuais e sua aceitação pelo FlexLab

Nome	Criador	Processador do *Host*	Processador do *Guest*	Sistema Operacional do *Host*	Sistemas Operacionais Suportados	Executa qualquer *SO?*	Método de Operação	Licença	Aceitável para o FlexLab?	Desempenho Relativo do *Guest*
Bochs	Kevin Lawton	Intel x86, AMD64, SPARC, PowerPC, Alpha, MIPS	Intel x86, AMD64	Windows, Linux, OS X, IRIX, AIX, BeOS	DOS, Windows, xBSD, Linux	Sim	emulação	LGPL	Não	Muito lento
Colinux	Dan Aloni helped by outros developers 1	Intel x86, outros?	Mesmo do *Host*	Windows NT (NT, 2000, XP, Server 2003), Linux?	Linux	Não	Porte de Sistema	GPL versão 2	Não	Nativo
Denali	University of Washington	Intel x86	Intel x86	Denali	Ilwaco, NetBSD	Não	Paravirtualização e Porte de Sistema	*?*	Não	Lento
DOSBox	Peter Veenstra e Sjoerd with community help	Intel x86, AMD64, SPARC, PowerPC, Alpha, MIPS	Intel x86	GNU/Linux, Windows, Mac OS Classic, Mac OS X, BeOS, FreeBSD, OpenBSD, Solaris, QNX, IRIX	Shell Emulado do DOS	Não	Emulação usando Tradução Dinâmica	GPL	Não	Extremamente Lento - porém a velocidade é irrelevante para o tipo de aplicação
DOSEMU	Community Project	Intel x86	Intel x86	Linux	DOS	Sim	Virtualização de *Hardware*	GPL versão 2	Não	Nativo
FreeVPS	PSoft	Intel x86, AMD64	Mesmo do *Host*	Linux	Várias Distribuições Linux	Não	Virtualização em Nível de SO	GPL versão 2	Não	Nativo
Integrity Virtual Machine	Hewlett-Packard	Itanium	Itanium	HP-UX	HP-UX (Linux, Windows, OpenVMS)	Sim	Virtualização	Proprietário	Não	Perto do Nativo
Jail	FreeBSD	Intel x86,	Mesmo do *Host*	FreeBSD	FreeBSD	Não	Virtualização em Nível de SO	FreeBSD	Não	Nativo
KVM	KVM	Processador Intel com extensões VT	Processador Intel com extensões VT	Linux	Linux	Não	Virtualização em Nível de SO	GPL2	Não	Nativo
Linux-VServer	Community Project	Intel x86, AMD64, IA-64, Alpha, PowerPC/64, PA-RISC/64, SPARC/64, ARM, S/390, SH/66, MIPS	Mesmo do *Host*	Linux	Várias Distribuições Linux	Não	Virtualização em Nível de SO	GPL versão 2	Não	Nativo
Mac on Linux	Mac On Linux [8]	PowerPC	PowerPC	Linux	Mac OS X, Mac OS 7.5.2 to 9.2.2, Linux	Não	Virtualização	GPL	Não	Nativo
OpenVZ	Projeto da Comunidade com suporte da SWsoft	Intel x86, AMD64, IA-64	Intel x86, AMD64, IA-64	Linux	Diversas Distribuições Linu	Não	Virtualização em Nível de SO	GPL	Não	Nativo
Parallels Workstation	Parallels, Inc.	Intel x86, Intel VT-x	Intel x86	Windows, Linux, Mac OS X (Versão Intel)	Windows, Linux, FreeBSD, OS/2, eComStation, MS-DOS, Solaris	Sim	Virtualização, Lightweight Hypervisor	Proprietário	Sim	Perto do Nativo
PearPC	Sebastian Biallas	x86, AMD64	PowerPC	Windows, Linux	OS X, Darwin, Linux	Sim	Emulação usando Tradução Dinâmica	GPL	Não	10% do Desempenho Nativo

Nome	Criador	Processador do *Host*	Processador do *Guest*	Sistema Operacional do *Host*	Sistemas Operacionais Suportados	Executa qualquer *SO?*	Método de Operação	Licença	Aceitável para o FlexLab?	Desempenho Relativo do *Guest*
QEMU	Fabrice Bellard com suporte de outros	Intel x86, AMD64, IA-64, PowerPC, Alpha, SPARC 32 e 64, ARM, S/390, M68k	Intel x86, AMD64, ARM, SPARC 32 e 64, PowerPC, MIPS	Windows, Linux, OS X, FreeBSD, BeOS	Windows, Linux, OS X, FreeBSD, BeOS e Outros	Sim	Recompliação Dinâmica	GPL/LGPL	Não	10 a 20% do Desempenho Nativo
QEMU c/ Módulo QKEMU	Fabrice Bellard	Intel x86, AMD86	O mesmo do *Host*	Linux, FreeBSD, Windows	Linux, FreeBSD, Windows	Sim	Virtualização	GPL versão 2	Sim	Perto do Nativo
QEMU c/ Módulo qvm86	Paul Brook	x86	x86	Linux, NetBSD, Windows	Linux, NetBSD, Windows	Sim	Virtualização	GPL	Sim	Perto do Nativo
SimNow	AMD	AMD64	AMD64	Linux (64bit), Windows (64bit)	Linux, Windows (32bit e 64bit)	Sim	*Cache* de código, Virtualização,	AMD Proprietário	Não	Lento
TRANGO	TRANGO Systems, GreNãoble, France	ARM, XScale, MIPS, PowerPC	ARM, MIPS e PowerPC Paravirtualizados	Nenhum	Linux, eCos, µC/OS-II	Sim	Paravirtualização e Porte de Sistema or Virtualização de *Hardware*	Proprietário	Não	Nativo
View-OS	Renzo Davoli e comunidade	Intel x86, PowerPC, AMD64 (in progress)	O mesmo do Host	Linux 2.6+	Linux	Não	Partial Virtualização por trapping usando syscall	GPL versão 2	Não	Perto do Nativo
User Mode Linux	Jeff Dike e Comunidade	Intel x86, outros?	O mesmo do Host	Linux	Linux	Não	Porte de Sistema	GPL versão 2	Não	Lento
VirtualBox	Sun Microsystems	x86, x86-64	x86	Windows, Linux, Mac OS X (Versão Intel), Solaris (x86-64)	DOS, Windows, Linux, FreeBSD, Solaris	Não	Virtualização	GPV versão 2	Sim	Perto do Nativo
VirtualPC 7 for Mac	Microsoft	PowerPC	Intel x86	OS X	Windows, OS/2, Linux	Sim	Recompilação Dinâmica	Proprietário	Não	Lento
VirtualLogix VLX	VirtualLogix	ARM, DSP C6000, Intel x86, Intel VT-x	O mesmo do Host	Nenhum: Instalação em baremetal	Linux, C5, VxWorks, Nucleus, DSP/BIOS e Proprietário OS	Sim	Paravirtualização e Porte de Sistema or Virtualização de *Hardware*	Proprietário	Não	Perto do Nativo
Virtual Server 2005 R2	Microsoft	Intel x86, AMD64	Intel x86	Windows 2003, XP	Windows NT, 2000, 2003, Linux (Red Hat e SUSE)	Sim	Virtualização (guest calls trapping)	Proprietário (Gratuito)	Não	Perto do Nativo
Virtuozzo	SWsoft	Intel x86, IA-64, AMD64	Intel x86, IA-64, AMD64	Linux & Windows	Várias Distribuições Linux e Windows	Não	Virtualização em Nível de SO	Proprietário	Não	Nativo
VMware ESX Server 3.0	VMware	Intel x86, AMD64	Intel x86, AMD64	Nenhum (instalação em BareMetal)	Windows, RedHat, SuSE, Netware, Solaris	Sim	Virtualização	Proprietário	Não	Perto do Nativo
VMware ESX Server 2.5.3	VMware	Intel x86, AMD64	Intel x86	Nenhum (instalação em BareMetal)	Windows, RedHat, SuSE, FreeBSD, Netware	Sim	Virtualização	Proprietário	Não	Perto do Nativo
VMware Server	VMware	Intel x86, AMD64	Intel x86, AMD64	Windows, Linux	DOS, Windows, Linux, FreeBSD, Netware, Solaris, Virtual Appliances	Sim	Virtualização	Proprietário (Gratuito)	Não	Perto do Nativo
VMware Workstation 5.5	VMware	Intel x86, AMD64	Intel x86, AMD64	Windows, Linux	DOS, Windows, Linux, FreeBSD, Netware,	Sim	Virtualização	Proprietário	Sim	Perto do Nativo

Nome	Criador	Processador do *Host*	Processador do *Guest*	Sistema Operacional do *Host*	Sistemas Operacionais Suportados	Executa qualquer *SO?*	Método de Operação	Licença	Aceitável para o FlexLab?	Desempenho Relativo do *Guest*
VMware Player	VMware	Intel x86, AMD64	Intel x86, AMD64	Windows, Linux	DOS, Windows, Linux, FreeBSD, Netware, Solaris, Virtual Appliances	Sim	Virtualização	Proprietário (Gratuito)	Sim	Perto do Nativo
Xen	University of Cambridge, Intel, AMD	Intel x86, AMD64, (PowerPC e IA-64 em desenvolvimentio)	O mesmo do Host	NetBSD, Linux	Linux, NetBSD, FreeBSD, OpenBSD, Windows XP & 2003 Server	Sim	Paravirtualização e Porte de Sistema ou Virtualização de *Hardware*	GPL	Sim	Nativo (Processador c/ suporte a Paravirtualização)
z/VM	IBM	z/Architecture	z/Architecture e predecessors	z/VM	Linux Proprietário	Sim	Virtualização	Proprietário	Não	Nativo
Zones	Sun Microsystems OpenSolaris	Intel x86, AMD64, UltraSPARC, SPARC64	O mesmo do Host	Solaris	Solaris, Linux (BreZ)	Não	Virtualização em Nível de SO	CDDL (Free)	Não	Perto do Nativo