Uma Visão Geral Sobre Sistemas Virtualizados

Rodolfo Wottrich[*]
RA 134077
Instituto de Computação (IC)
Universidade Estadual de Campinas (UNICAMP)

Thiago Genez[†]
RA 100616
Instituto de Computação (IC)
Universidade Estadual de Campinas (UNICAMP)

Walisson Pereira[‡]
RA 115168
Instituto de Computação (IC)
Universidade Estadual de Campinas (UNICAMP)

RESUMO

Em computação, o termo *virtualização* é um conceito amplo, que pode ser definido como a capacidade de estender ou substituir um recurso (software ou hardware), de modo a imitar um comportamento. Este trabalho tem como objetivo apresentar uma visão geral dos aspectos relacionados ao uso de sistemas computacionais virtualizados, incluindo os principais conceitos básicos, tipos de máquinas virtuais, formas de virtualização e a virtualização da arquitetura $x86$. Além disso, apresentamos as principais vantagens e desvantagens do uso da virtualização e a tendência dela na computação em nuvem. Com este trabalho, mostramos que a virtualização é uma técnica muito importante para o estado atual da computação e está presente em diversas aplicações.

Categorias e Descritores de Conteúdo

C.0 [**Computer Systems Organizations**]: General—*Hardware/Software Interfaces*

Termos Gerais

Arquitetura de Computadores, Virtualização

Palavras-chave

Virtualização, *Hypervisor*, Máquinas virtuais, Processadores

1. INTRODUÇÃO

O termo genérico *virtualização* é utilizado, hoje em dia, para referenciar abstração dos recursos de um computador. Em outras palavras, virtualização é geralmente referenciada como uma *camada de software* que abstrai as características físicas do hardware, fornecendo recursos virtualizados (de *hardware*) para as aplicações de alto nível. Assim, através da virtualização, uma máquina física é, de modo geral, logicamente dividida em um monitor de máquina virtual – *Virtual Machine Monitor* (VMM), também conhecido como *hypervisor* – e em várias máquinas virtuais – *Virtual Machine*s (VMs) [12]. Além disso, cada máquina virtual pode ser independente, podendo ter seus próprios aplicativos, serviços e Sistema Operacional (SO); ou seja, cada máquina virtual atua como se estivesse instalado isoladamente em uma máquina física. O *hypervisor*, por sua vez, tem que gerenciar essas VMs e fornecer recursos computacionais virtualizados.

Neste trabalho, vamos apresentar uma visão geral sobre sistemas virtualizados, incluindo interfaces entre hardware e software. O restante deste trabalho esta organizado da seguinte maneira. A Seção 2 descreve um breve histórico sobre a virtualização que iniciou na década de 60. Na sequência, a Seção 3 apresenta os conceitos básicos, incluindo diferenças entre abstração e virtualização, tipos de máquinas virtuais, tipos de *hypervisores*, teorema de Popek & Goldberg e maneira de realizar a virtualização. A virtualização da arquitetura $x86$ é apresentada na Seção 4, enquanto vantagens e desvantagens de sistemas virtualizados são descritos na Seção 5. A Seção 6 relata as principais tecnologias de virtualização de código-aberto, enquanto as tendências sobre sistemas virtualizados são descritos na Seção 7. A Seção 8 finaliza este trabalho com as considerações finais.

2. UM BREVE HISTÓRICO

O conceito de *máquina virtual* não é recente. Em meados da década de 60, o centro de pesquisa da IBM, usando a máquina IBM 7044, fez uma imagem do 7044 em cada partição do sistema. Cada imagem foi chamada de 7044/44X e foi criada para que a IBM entendesse melhor os sistemas operacionais multiprogramados. Esse foi o início da noção da IBM de um máquina virtual, uma cópia de uma máquina atual em um espaço reduzido de memória [8]. No final da década, um grupo da IBM construíu um sistema operacional e nomeou-o the *Cambridge Monitoring System* (CMS). O CMS foi um experimento em sistemas *time sharing* e finalmente tornou-se a arquitetura usada para a máquina VM/370, que foi vendida como um sistema *time sharing*. Várias empresas e universidades usaram esse sistema operacional por permitir que seu poder de processamento e recursos disponíveis pudessem ser compartilhados entre vários usuários. Cada usuário tinha sua própria máquina virtual para trabalhar e, apesar de compartilhar recursos com todos os demais usuários, suas ações não afetavam os demais. Na década de 70, Popek & Goldberg formalizaram vários conceitos associa-

[*]rodolfo.wottrich@students.ic.unicamp.br.
[†]thiagogenez@ic.unicamp.br.
[‡]walisson@ic.unicamp.br.

dos às máquinas virtuais, e também definiram as condições necessárias para que uma plataforma de hardware suporte de forma eficiente a virtualização [17]. Na década de 80, a popularização do PC barateou o custo do hardware e, consequentemente, inibiu a importância da virtualização. [14] Desde a década de 70 já havia a ideia de criar programas de computador realmente portáteis, ou seja, programas que funcionem em qualquer plataforma. Esta ideia foi possível somente em meados da década de 90, utilizando a noção de máquinas virtuais para executar aplicações desenvolvidos em Java[1]. A principal característica da programação Java é o fato dela não gerar binários executáveis em arquiteturas ou sistemas operacionais específicos, mas um *bytecode* que deve ser executado pela *Java Virtual Machine* (JVM). Em outras palavras, um *bytecode* em Java pode rodar em qualquer arquitetura que tenha uma máquina JVM instalada. O mantra original do Java era "*write once, run anywhere*" (escreva uma vez, rode em qualquer lugar) [7]. Com o aumento de desempenho e funcionalidades do hardware PC e o surgimento da linguagem Java, o interesse pelas tecnologias de virtualização voltou à tona [14].

3. CONCEITOS BÁSICOS

Através da virtualização, é possível que um programa desenvolvido para uma plataforma A possa ser executada sobre uma plataforma distinta B. Nesta seção vamos descrever os principais conceitos sobre o funcionamento da virtualização.

3.1 Abstração e Virtualização

Antes de introduzirmos os conceitos básicos, temos que, primeiramente, distinguir os termos *abstração* e *virtualização*. Enquanto abstração consistem em providenciar uma interface simplificada e homogênea para acessar os recursos (hardware ou software), a virtualização cria novas interfaces a partir das existentes [22]. Por exemplo, os projetistas de hardware da AMD e Intel implementam o conjunto de instruções IA-32[2], enquanto os engenheiros de software da Microsoft desenvolvem aplicativos que são compilados para esse conjunto específico de instruções. Em outras palavras, como ambos grupos (projetistas de hardware e engenheiros de software) são independentes e compartilham o mesmo conjunto de instruções, é esperado que o software Microsoft (*e.g* com extensão *.exe*) executem corretamente em qualquer hardware IA-32. Essa interface bem-definida de conjunto de instruções é a camada de abstração. Entretanto, quando não há o compartilhamento do mesmo conjunto de instruções, haverá, na maioria das vezes, conflito de execução; ou seja, o software não irá executar no hardware. É nesse instante que entra em cena a virtualização, proporcionando uma camada de compatibilidade entre (i) software e hardware e (ii) entre software. Nas próximas seções vamos apresentar melhores detalhes sobre virtualização.

3.2 Interfaces de Sistema

De modo geral, computadores são utilizados através de três grandes componentes [6]: hardware, sistema operacional[3] e aplicações (ou processos[4].). Basicamente, o hardware executa as operações solicitadas pelas aplicações, através do sistema operacional. Este, por sua vez, recebe essas solicitações através das chamadas de sistema (*syscalls*)[5] via interrupções[6] e, além disso, controla o acesso ao hardware; essencialmente quando os recursos físicos são compartilhados, como, por exemplo, memória, discos e dispositivos de entrada/saída. Como dito na Seção 3.1, os computadores são geralmente caracterizados por níveis de abstração entre software e hardware. Dessa forma, os SOs oferecem uma visão abstrata (e de alto nível) dos recursos de hardware, a fim de facilitar o uso e desacoplar dependências tecnológicas subjacentes. Assim, abstrações fornecidas pelo SO às aplicações são desenvolvidas incrementalmente e em níveis, os quais são separados por interfaces bem-definidas [6, 10]. Portanto, cada interface encapsula as abstrações dos níveis inferiores, facilitando a evolução dos sistemas de computadores de forma geral. Conforme ilustra a Figura 1, as interfaces mais utilizadas entre os componentes de computação são o conjunto de instruções – *Instruction Set Architecture* (ISA), chamadas de sistemas (*system calls*) e chamadas de bibliotecas (*libcalls*).

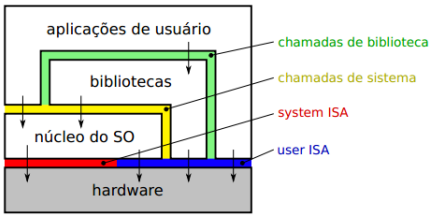

Figure 1: Componentes e interfaces de um SO [6].

Conjunto de instruções (ISA): Arquitetura do conjunto de instruções, ou código de máquina, aceito pelo processador, incluindo as operações de acesso físico aos dispositivos de hardware, tais como: memória, portas de entrada/saída e relógio do sistema. Em outras palavras, é a interface básica entre hardware e software. A ISA é dividida em duas partes: *instruções de usuário (user ISA)* e *instruções de sistema (system ISA)*. A primeira está relacionada com as instruções do processador que são executadas em modo não-privilegiado[7] (espaço de usuário), instruções que não oferecem risco ao SO. Enquanto a segunda compreende as instruções do processador que são executadas em modo privilegiado (espaço de núcleo), instruções que só devem ser executadas pelo núcleo do SO ou sob sua supervisão.

Chamadas de sistema (*syscalls*): Instruções que são invocadas pelos processos através de interrupções que, como

[1] `http://www.java.com/`
[2] *Intel Architecture* 32–bit
[3] Em tempos modernos, os SOs são considerados *multitarefa*, pois permitem a coexistência de múltiplos processos simultâneos.
[4] O termo *processo* é geralmente usado para designar uma programa em execução
[5] O termo *chamada de sistema* é o mecanismo utilizado pelos processos para requisitar um serviço do núcleo do SO.
[6] O termo *interrupção* é um sinal de indica uma *troca de contexto*, ou seja, o processador para de fazer o que está fazendo para atender o dispositivo que pediu a interrupção.
[7] Conjunto de instruções que não podem ser fornecidos livremente para as aplicações, pois o seu uso indevido poder causar problemas à integridade do SO. Por exemplo, acesso ao disco (um recurso compartilhado); o acesso indiscriminado a qualquer área do disco pode comprometer a segurança e integridade do sistema de arquivos.

já descrito, são gerenciadas pelo sistema operacional. Portanto, as chamadas de sistema possibilitam um acesso controlado das aplicações aos recursos compartilhados (memória, disco, etc.) e às instruções privilegiadas do processador.

Chamadas de bibliotecas (*libcalls*): Coleção de subprogramas que encapsulam chamadas de um sistema operacional, facilitando o desenvolvimento de aplicações para o próprio SO. Note que a biblioteca deve ser compilada para a arquitetura na qual as aplicações serão executadas.

3.3 Virtualização e Máquinas Virtuais

Com o surgimento de várias arquiteturas de hardware e sistemas operacionais, as aplicações escritas para uma certa plataforma operacional[8] em geral não funcionam em outras plataformas, devido à incompatibilidade tanto do hardware (ISA) quanto do sistema operacional (software). A Figura 2 ilustra essa situação. No exemplo, verificamos não ser possível executar diretamente na arquitetura *Scalable Processor Architecture* (SPARC) uma aplicação compilada para a arquitetura $x86$, pois as instruções em linguagem de máquina do programa não são compreendidas pelo processador SPARC. Aliás, também não é possível executar diretamente no Windows um software desenvolvido para o sistema GNU/Linux, pois as bibliotecas de desenvolvimento (*libcalls*) são compiladas baseada no sistema operacional.

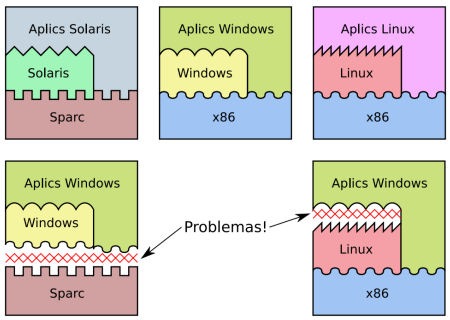

Figure 2: Problemas de compatibilidade entre interfaces [9].

Uma das maneiras de solucionar a incompatibilidade entre plataformas operacionais é através da tecnologia de virtualização. Em outras palavras, a virtualização "cria" uma camada de software para comunicar (e acoplar) diferentes plataformas que são executadas simultaneamente em uma mesma máquina física. Essa camada é denominada de *monitor de máquina virtual* (VMM) ou *hypervisor*, o qual é ilustrado na Figura 3. Imediatamente acima do *hypervisor* estão situadas as máquinas virtuais, as quais podem ser definidas como *uma cópia eficiente e isolada de uma máquina real*, segundo Popek & Goldberg em [17]. Aliás, o VMM é responsável pelo controle e virtualização dos recursos físicos compartilhados entre as máquinas virtuais, como por exemplo processadores, memória RAM, disco rígido e dispositivos de entrada e saída. Portanto, um computador físico hospedeiro (denominado *host*) pode compartilhar seus recursos de hardware entre múltiplas máquinas virtuais convidadas (denominada *guest*), as quais são gerenciadas pelo *hypervisor*. Além disso, as máquinas virtuais são, de modo geral, independentes uma das outras e, em alguns casos, atuam como se tivessem sido instaladas isoladamente no computador hospedeiro [24].

Figure 3: Acoplamento entre interfaces distintas [9].

3.3.1 Tipos de Máquinas Virtuais

De acordo com o tipo de sistema convidado suportado, as máquinas virtuais podem, segundo [10], ser classificadas em duas categorias: VM de aplicação e VM de sistema. As VMs de aplicação são máquinas virtuais que suportam apenas uma aplicação específica. Por exemplo, a máquina virtual *Java* – JVM – é um software que carrega e executa os aplicativos escritos em *Java*, através da conversão dos *bytecodes*[9] em código executável de máquina onde a JVM está sendo executada. Portanto os aplicativos em Java são independentes da plataforma operacional, executando em qualquer sistema que simplesmente possua uma JVM. Já as VMs de sistema são máquinas virtuais que suportam sistemas operacionais completos, com seus aplicativos e serviços. O *VMware*[10] e o *Virtual Box*[11] são exemplos dessa categoria. A Figura 4 exemplifica essas duas categorias.

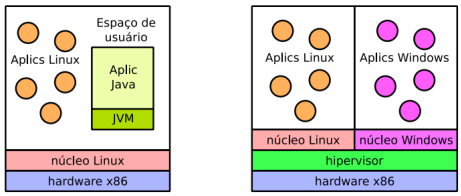

Figure 4: VMs de aplicação (esq.) e de sistema (dir.) [10].

3.3.2 Tipos de Hypervisor

Existem basicamente duas abordagens de *hypervisor* de sistema [10]: nativos (ou de tipo I) e convidados (ou de tipo II). Na primeira abordagem, o *hypervisor* é implementado diretamente sobre o hardware da máquina real, sem a necessidade de um sistema operacional subjacente; enquanto na segunda, o *hypervisor* é executado como se fosse um processo normal sobre um sistema operacional nativo (anfitrião) subjacente. A Figura 5 ilustra essas duas abordagens de *hypervisor*. Segundo [10], um *hypervisor* convidado suporta, de modo geral, apenas uma VM com uma instância de sistema operacional convidado, e se mais VMs forem necessárias, mais *hypervisors* (processos do sistema operacional nativo) devem ser executados. O *hypervisor* nativo garante um aumento no desempenho por ter menos intermediários entre o software em execução na VM e o hardware da máquina real, mas exige que esse *hypervisor* seja mais complexo, utilizando técnicas de emulação e simulação para oferecer à máquina virtual um contexto similar ao de uma máquina real. Por

[8] Neste trabalho o termo *plataforma operacional* refere-se ao conjunto formado pelo hardware, SO e aplicações.

[9] Código intermediário entre o código-fonte e o binário.

[10] `http://www.vmware.com/`

[11] `https://www.virtualbox.org/`

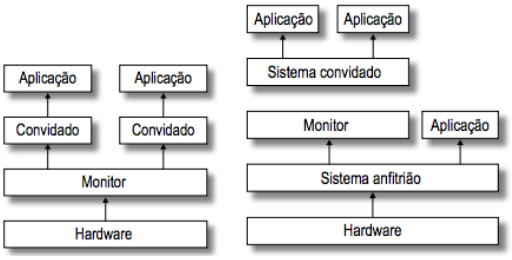

Figure 5: VMMs nativo (esq.) e convidado (dir.) [10].

outro lado, a implementação do *hypervisor* é mais simples, pois o sistema operacional subjacente oferece uma série de primitivas de acesso ao hardware e controle de periféricos. Essas duas abordagens, entretanto, são teóricas e na prática são implementadas melhorias de desempenho no *hypervisor* e nas VMs para otimizar o acesso ao hardware.

3.3.3 Anéis de Proteção da CPU

Os processadores atuais utilizam a noção de *anéis de proteção ou domínio de proteção* para controlar o acesso ao conjunto de instruções e aos recursos físicos do sistema [15]. Em outras palavras, os anéis determinam níveis de privilégios da execução de um código para proteger o hardware (e periféricos) de falhas e comportamentos maliciosos. Por exemplo, um código executado no nível 0 (conhecido como anel central) possui acesso completo ao hardware, enquanto um código executado em um nível $k > 0$ (anel externo) possui menos privilégios. Isto é, quanto mais distante um anel se encontra do anel central, menor será o seu nível de privilégio. A arquitetura $x86$ suporta 4 anéis de proteção (ver Figura 6), sendo o anel 0 definido para o *espaço de kernel* e o anel 3 estipulado *espaço de usuário*. Portanto, dependendo da forma de virtualização, podemos descrever em qual anel o *hypervisor* e as VMs estarão situados.

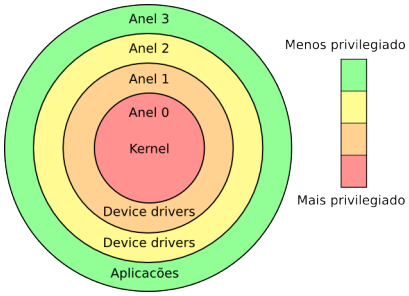

Figure 6: Anéis de privilégios da arquitetura $x86$

A Figura 7 ilustra a noção de anéis em um computador com *hypervisor* nativo, onde este executa no anel 0, enquanto os SOs hóspedes (ou convidados) no anel 1 (ou 2) e as aplicações continuam intactas, no anel 3. Isto simplifica o tratamento de interrupções pelo *hypervisor*, uma vez que é possível distinguir exceções e interrupções das aplicações do nível 3 das dos SOs hóspedes no nível 2.

3.3.4 Teorema de Popek & Goldberg

Em 1974, período onde foram desenvolvidas as primeiras soluções de virtualização, Popek & Goldberg descreveram em [17] um teorema para que uma arquitetura de computador possa implementar um *hypervisor* de forma eficiente e,

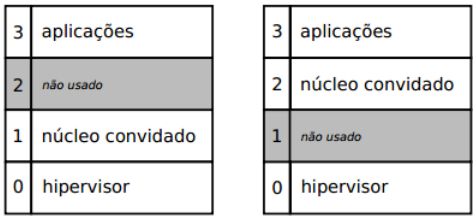

Figure 7: Virtualização com modelo 0/1/3 (esq.) e 0/2/3 (dir.) [6]

consequentemente, suportar VMs de forma adequada. Esses teoremas se baseam na classificação das instruções de máquinas, as quais são: privilegiadas e sensíveis. Além disso, também supõem que o processador possui modos de operações *privilegiado* e *usuário* (ver Seção 3.3.3), funcionalidade existente na maioria das arquiteturas desde a *IBM* 370, inclusive a $x86$. Uma instrução de máquina é dita *privilegiada* se a sua execução em modo usuário gera uma exceção, ou seja, é interrompida e o sistema operacional é notificado. As instruções *sensíveis* são aquelas que podem podem consultar ou alterar o *status* do processador, isto é, os registradores que armazenam o *status* atual da execução no hardware.

O teorema de Popek & Goldberg descreve que o *hypervisor* pode ser implementado em uma arquitetura de computador em que as instruções sensíveis sejam um subconjunto das instruções privilegiadas. Segundo esse teorema, toda instrução sensível deve ser também privilegiada. Assim, quando uma instrução sensível for executada por uma aplicação não-privilegiada (núcleo do sistema convidado, por exemplo), irá provocar uma interrupção que deverá ser interceptada e tratada pelo *hypervisor*. Este, por sua vez, irá simular (ou emular) o efeito desejado da instrução sensível de acordo com o contexto de onde a instrução sensível foi executada (se foi de uma das VMs monitoradas ou pelo próprio *hipervisor*); garantindo, portanto, o isolamento entre VMs. Note que, como todas as instruções não-privilegiadas não são potencialmente prejudiciais à virtualização, então elas podem ser diretamente executadas pelo processador, garantindo, então, que o desempenho de uma máquina virtualizada seja muito próximo de uma máquina real. Entretanto, existem algumas arquiteturas (assim como a $x86$) que não respeitam o teorema de Popek & Goldberg. Assim, podem existir instruções sensíveis que executem sem gerar interrupções; impedindo, então, o *hypervisor* de interceptá-las e interpretá-las. Portanto, houve a necessidade de se criar artifícios para contornar esse problema, dos quais os dois predominantes são a tradução binária e a paravirtualização. Ambas técnicas são discutidas na Seção 3.4.

3.4 Formas de Virtualização

Conforme dito anteriormente, a virtualização é a tecnologia capaz de abstrair as configurações do hardware via software, como por exemplo, fazer os recursos físicos parecerem diferentes do que realmente são. Graças à essa tecnologia, é possível, portanto, dividir os recursos de um computador em vários ambientes de execução (plataforma operacional). Entretanto, dependendo de como o *hypervisor*, o hardware físico do hospedeiro e as VMs se interagem entre si, podemos elencar várias formas de virtualização; detalharemos brevemente as principais nesta seção.

3.4.1 Virtualização Completa ou Total

Na virtualização completa, o *hypervisor*, situado no anel 0 da CPU, disponibiliza uma réplica (emulação completa) do hardware subjacente, ou seja, a interface de acesso ao hardware é virtualizada (incluindo todas as instruções do processador e os dispositivos de entrada e saída). Alguns autores denominam esta virtualização como *virtualização do hardware* [1, 13, 16, 19]. Dessa maneira, o sistema operacional visitante não precisa sofrer modificações para ser executado na máquina virtual [13]. Além disso, esse tipo de virtualização utiliza a técnica de *tradução binária*, onde o *hypervisor* analisa, reorganiza e traduz as seqüências de instruções (binários) emitidas pelo SO convidado em novas seqüências de instruções [9]. É importante frisar que esta análise deve ser feita em tempo de execução e antes que o binário seja enviado para execução na máquina real. Dessa forma, através da tradução binária, é possível (i) adaptar as instruções geradas pelo sistema convidado à interface de conjunto de instruções (ISA) do sistema real, caso não sejam idênticas; (ii) detectar e tratar instruções sensíveis não-privilegiadas que não geram interrupções ao serem invocadas pelo SO convidado (uma das técnicas usadas para virtualizar a $x86$) e (iii) otimizar as seqüências de instruções geradas pelo SO convidado, a fim de melhorar o desempenho de sua execução.

A principal vantagem dessa técnica de virtualização é o fato de que o sistema operacional virtualizado não precisa de qualquer tipo de alteração. Em contrapartida, a técnica de tradução binária acrescenta uma complexidade considerável ao *hypervisor*, reduzindo o desempenho da virtualização em até 30%, quando comparado com a execução direta na máquina real [21]. Aliás, Laureano *et al.* afirmam em [9, 10] que o *hypervisor* desse tipo de virtualização pode ser nativo, enquanto Sahoo *et al.* descrevem em [21] que o *hypervisor* pode ser convidado. *VMWare*, *VirtualBox* e *QEmu*[12] são alguns exemplos de *hypervisors* que aplica a virtualização completa. A virtualização completa só foi possível em 1972, na série *IBM System*/370, após a adição do mecanismo de memória virtual. Na arquitetura $x86$, a virtualização total foi possível apenas em 2005–2006 quando foi incorporado a técnica *Hardware-assisted virtualization* em sua arquitetura, passando a obedecer o teorema de Popek & Goldberg. A Seção 4 descreve sobre a virtualização da $x86$.

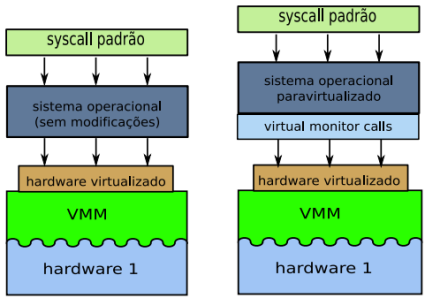

Figure 8: Virtualização total (esq.) e paravirtualização (dir.) [10]

3.4.2 Paravirtualização

Na paravirtualização, o sistema operacional convidado é modificado para invocar o *hypervisor* sempre que executar uma instrução que altere o estado do hardware, ou seja, uma instrução sensível. Assim, a técnica de *paravirtualização* recebeu este nome pois a máquina virtual tem "consciência" de que está sendo executada em um ambiente virtualizado. Dessa forma, o *hypervisor* não precisa analisar e testar os binários do SO convidado, o que representa um ganho significativo de desempenho. Portanto, as modificações na interface de sistema do hardware virtual (instruções de sistema) exigem uma adaptação no *kernel* dos SOs convidados, para que estes possam executar sobre o hardware virtual. Por outro lado, a interface de usuário do hardware virtual (instruções de usuário) é mantida, isto é, permite que as aplicações convidadas executem sem alterações de código. A Figura 8 mostra a comparação, de modo geral, entre a virtualização total e a paravirtualização. Uma das vantagens da paravirtualização é a possibilidade do SO convidado acessar alguns recursos de hardware diretamente, ou seja, sem a interferência ativa do *hypervisor*. Porém, esse acesso é apenas monitorado pelo *hypervisor*, informando o SO convidado de seus limites, como por exemplo, áreas de disco e memória disponíveis [10]. Por outro lado, a principal desvantagem desta virtualização é a necessidade de se alterar o *kernel* do SO convidado, o qual nem sempre é *open-source*. Por exemplo, para portar o Linux foi preciso modificar 2995 linhas de código (aproximadamente $1,36\%$ do código fonte total). Em contrapartida, para o Windows XP foi necessário alterar 4620 linhas de código (aproximadamente $0,04\%$ do código fonte total) [2]. Entretanto, o Windows XP modificado não pode ser distribuído por questões de licença. O maior representante desta técnica de virtualização é o *hypervisor* Xen[13].

3.4.3 Virtualização de Interface de Sistema

Neste tipo de virtualização, apenas as instruções de sistema (*system ISA*) – instruções privilegiadas – são virtualizadas, mantendo o conjunto de instruções do usuário (*user ISA*) intacta. Dessa forma, as instruções não-sensíveis (aplicações convidadas, por exemplo) podem ser executadas diretamente pelo processador da máquina real, sem perda de desempenho. Aliás, esse tipo de virtualização é mais eficiente que as anteriores, pois o *hypervisor* precisa emular apenas as instruções sensíveis do processador virtual, as quais são executadas em modo privilegiado pelo SO convidado [3]. A desvantagem desta abordagem é que somente os SOs e aplicações convidados desenvolvidos para o processador real podem ser executados. A Figura 9 mostra a comparação entre a virtualização de interface de sistema com a completa.

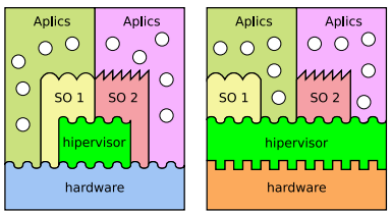

Figure 9: Virtualização de interface de sistema (esq.) e completa (dir.) [9]

4. VIRTUALIZAÇÃO EM X86

A arquitetura $x86$, uma das arquiteturas de computadores mais utilizadas hoje em dia, não foi projetada, a princípio, para ser virtualizada [13]. A grande dificuldade de se imple-

[12] http://wiki.qemu.org/Main_Page

[13] http://xen.org/

mentar um *hypervisor* na arquitetura $x86$[14] é o fato que existem pelo menos 17 instruções sensíveis que não são privilegiadas, ou seja, cuja execução em modo usuário não gera uma exceção, mas é tratada pelo próprio processador ($x86$) [18]. Em outras palavras, a arquitetura $x86$ viola o teorema de Popek & Goldberg, descrito na Seção 3.3.4. A questão é que em uma máquina real que dê suporte à diversas máquinas virtuais (virtualização completa, por exemplo), o *hypervisor* deve executar em modo privilegiado do processador (anel 0), visto que será ele quem gerenciará os recursos do sistema. Entretanto, o sistema operacional é o único software que executa em modo privilegiado, de forma a gerenciar os recursos físicos do sistema. Assim, quando a $x86$ for virtualizada, obviamente não será executada em modo privilegiado (e sim no anel 1 ou 2, ver Figura 7); mas, como a virtualização deve ser transparente para o SO, ele irá executar suas instruções como se estivesse no modo privilegiado. Portanto, quando o SO executar algumas destas 17 operações ditas "críticas", elas serão abortadas, enquanto idealmente deveria ser gerada uma exceção a ser tratada pelo *hypervisor*. Assim, o desafio na implementação de *hypervisor* na arquitetura $x86$ é a forma de lidar com instruções críticas. Os fabricantes de processadores $x86$, AMD e Intel desenvolveram extensões para a arquitetura x86 para suportarem a virtualização sem violar o teorema de Popek & Goldberg, através da técnica denominada *virtualização assistida pelo hardware* [13] ou também conhecida como *virtualização com suporte em hardware* [11], o qual está descrito na próxima seção.

Virtualização Assistida pelo Hardware. Com o desenvolvimento e os resultados interessantes de eficiência obtidos por soluções de paravirtualização e virtualização completa, os produtores de hardware $x86$, no qual se destacam a Intel e a AMD, passaram a se importar com este ramo de mercado. Como resultado, a primeira série de processadores $x86$ com *suporte a virtualização* foi lançada quase simultaneamente pelas duas empresas: o Intel *VT-x*[15] (de codinome *Vanderpool*) em 2005 e o *AMD-V*[16] (de codinome *Pacifica*) em 2006 [10, 13, 23]. Estas linhas de processadores resolveram o problema das instruções críticas, modificando o conjunto de instruções de máquina da $x86$ a fim de obedecer, então, ao teorema de Popek & Goldberg. Note que, as soluções da Intel e da AMD foram desenvolvidas independentemente uma da outra e, além disso, são incompatíveis, mas atingem o mesmo propósito. Resumidamente, a ideia de ambas tecnologias é definir dois modos possíveis de operação do processador: *root* e *non-root* [6, 9]. O primeiro modo é destinado à execução do *hypervisor* e equivale ao funcionamento de um processador convencional, enquanto o segundo modo é destinado à execução das máquinas virtuais. Aliás, ambos modos suportam os 4 anéis de privilégio, possibilitando executar os SOs convidados sem modificá-los. Além disso, dois procedimentos de transição entre os modos são definidos: VM *entry* (transição *root* → *non-root*) e VM *exit* (transição *non-root* → *root*). Assim, para gerenciar o estado do processador (conteúdo dos registradores) é definida uma *Virtual-Machine Control Structure* (VMCS), uma estrutura de dados que contém duas áreas: uma para os SOs convidados e outra para o *hypervisor*.

Esses procedimentos funcionam da seguinte maneira: na transição VM *entry*, o estado do processador é salvo na área de *hypervisor* da VMCS e, em seguida, o estado do processador é lido a partir da área de SOs convidados da VMCS; enquanto a transição VM *exit* faz com que o estado do processador seja salvo na área de SOs convidados e o estado anterior do *hypervisor* é restaurado. A Figura 10 mostra uma visão geral da arquitetura Intel VT. As instruções sensíveis e as interrupções (geradas dentro da VM, isto é, em modo *non-root*) provocam a transição VM *exit*, devolvendo o processador ao *hypervisor* em modo *root*. Note que ambos procedimentos são gerenciados e configurados pelo próprio *hypervisor*. Portanto, o suporte à virtualização da $x86$ foi possível através do procedimento chamado de *ring deprivileging*, determinando o SO convidado a executar no anel 1 ou 2 de forma transparente, mas ele "acredita" que está sendo executando no anel 0. Isto é, o SO operará em um anel 0 fictício e não terá conhecimento da presença do *hypervisor* em um nível de privilégio superior.

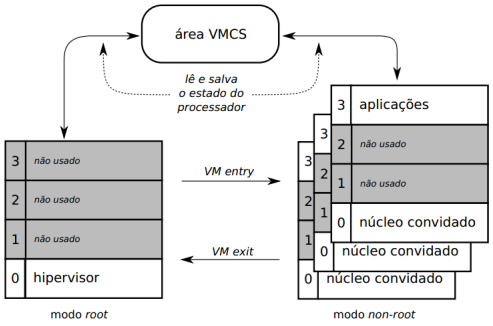

Figure 10: Visão geral da arquitetura Intel VT [6]

5. VANTAGENS E DESVANTAGENS DE SISTEMAS VIRTUALIZADOS

As principais vantagens dos sistemas virtualizados são [21]:
Flexibilidade de alocação de recursos: é possível, em uma única máquina física, ter várias VMs em execução, alocando os recursos de hardware sob demanda e até mesmo migrar serviços entre VMs, simplesmente ajustando os recursos alocados pelo *hypervisor*.
Disponibilidade: cada sistema é independente. Logo, uma queda de serviço permite a reinicialização ou simplesmente a ativação de uma nova máquina virtual em instantes.
Escalabilidade: é fácil adicionar ou remover novas instâncias da máquina virtual. Caso a demanda por capacidade cresça com o tempo, é fácil inserir um nó físico com a instalação básica do *cluster* e pô-lo para executar mais VMs.
Uso eficiente do hardware: o uso de várias máquinas virtuais permite o uso mais eficiente dos recursos computacionais da máquina anfitriã (hospedeira).
Segurança: cada sistema executa isoladamente e funciona de forma independente dentro de seu espaço alocado no VMM. Caso uma aplicação tenha algum problema, ela não afetará as demais máquinas virtuais.
Custos: é possível reduzir custos monetários através da instanciação de pequenos servidores virtuais dentro de servidores com alto poder computacional.
Aplicações legadas: se uma organização decide migrar

[14]Processadores Intel Pentium IV (e anteriores).
[15]Se aplica às arquitetura $x86$ de 32 e 64 bits, como o *Pentium D*, família *Core 2 Duo* e família Intel *Core i*$\{5,7\}$.
[16]Se aplica às arquiteturas $x86$ de 64 bits como o *Athlon*, *Turion*, *Phenom* e as linhas mais recentes [6].

para um SO diferente, é possível executar aplicações antigas nos antigos SOs como um sistema convidado de uma VM; além de reduz custos monetários com migração de hardware.

As principais desvantagens são [21]:
Sobrecarga: o sistema que hospeda uma ou várias máquinas virtuais acaba perdendo desempenho devido ao compartilhamento de seus recursos.
***Single point of failure* (SPOF)**: mesmo a VM sendo desacoplada do hardware, ela ainda depende das operações do hardware. Portanto, falhas no hardware levarão a falhas de todas as VMs suportadas.
Interface de administração: está intimamente ligada à plataforma de virtualização. Isto pode ser um problema que dificulta o uso de várias plataformas no mesmo ambiente.

6. TECNOLOGIAS DE VIRTUALIZAÇÃO

Nesta seção vamos descrever alguns exemplos de tecnologias de máquinas virtuais de código-aberto.

6.1 XEN

Como dito anteriormente, na Seção 3.4.2, o Xen é baseado na paravirtualização; assim, os SOs convidados devem ter ciência de que estão sendo operados sobre o núcleo Xen, evocando-o explicitamente para a realização das operações sensíveis [2]. O Xen não possui *drivers* de dispositivos e, com isso, não é possível rodar um SO convidado diretamente nele. Uma VM, instanciada no *domínio* 0, é invocada para fazer a comunicação entre o Xen e os sistemas hóspedes. Em outras palavras, a VM do domínio 0 roda um núcleo Linux modificado e possui privilégios para acessar os dispositivos de entrada/saída e as demais VMs, as quais são instanciadas no *domínio U* (não-privilegiadas) e executam os SOs convidados. Cada VM do domínio U é criada, inicializada e desligada através do domínio 0, e também possui *drivers* virtuais para acessar os recursos de hardware. Além de possuir os *drivers* dos dispositivos da máquina física, a VM do domínio 0 tem 2 *drivers* que tratam requisições de acessos à rede e aos discos realizadas pelas VMs do domínio U. Portanto, só a VM do domínio 0 tem acesso direto aos recursos da máquina física, enquanto que as demais VMs (domínio U) têm acesso a uma abstração dos recursos, os quais para serem acessados, têm que passar pelo domínio 0 [25].

6.2 KVM

O *Kernel-based Virtual Machine* (KVM)[17], é uma solução de virtualização completa que se aproveita de toda estrutura de *drivers* já existente no *kernel* para ter acesso ao hardware. Para seu funcionamento, é necessário possuir um processador com suporte a virtualização (*Intel* VT ou AMD-V). Através do KVM, é possível rodar múltiplas VMs que rodem imagens Linux ou Windows, sem a necessidade de modificá-los. Cada VM tem seu hardware virtualizado privado como placa de rede, disco, adaptador gráfico, etc. O KVM está contido no *kernel* do Linux desde a versão 2.6.20.

6.3 VirtualBox

O VirtualBox [18] emula completamente o hardware e não impõe a necessidade de se ter um processador com suporte a virtualização para executar uma máquina virtual. Caso o processador possua suporte a virtualização, então esse suporte é utilizado. Uma outra característica do VirtualBox é possuir uma interface intuitiva.

[17]http://www.linux-kvm.org
[18]https://www.virtualbox.org

7. TENDÊNCIAS

Nesta seção vamos apresentar algumas tendências sobre os sistemas virtualizados encontrados na literatura, das quais o uso mais recente se encontra na computação em nuvem [26].

7.1 Provedores de Infraestrutura

Em *datacenters*, os administradores poderão facilmente gerenciar e monitorar milhares de VMs executando em centenas de máquinas reais, tudo a partir de um só terminal [20]. Ao invés de configurar computadores individuais, é possível simplesmente instanciar novas VMs a partir de modelos pré-definidos e monitorar os recursos utilizados, tornando o mapeamento dos recursos físicos uma tarefa altamente dinâmica. Esse cenário é um exemplo de um provedor de *Infrastructure-as-a-Service* (IaaS) da computação em nuvem[19], que fornece VM como serviço através do modelo de tarifação *pago-pelo-uso*. Um desafio enfrentado atualmente por esses provedores é o *hot migration*, uma técnica onde o *hypervisor* tem que ser capaz de migrar suas VMs entre outras máquinas físicas do *datacenter*. O objetivo é a migração e a redistribuição de máquinas virtuais para minimizar o número de computadores físicos ativos enquanto os computadores ociosos são desligados, a fim de diminuir o consumo de energia elétrica. Esse paradigma é a base da *computação verde* e, portanto, contribui com a redução da emissão de dióxido de carbono (CO_2) na atmosfera e, consequentemente, do efeito estufa. Logo, para garantir um gerenciamento eficiente dos recursos e, ao mesmo tempo, proporcionar sua maior utilização, o Provedor de Infraestrutura tem que lidar com o equilíbrio entre os parâmetros energia e desempenho [4, 5].

7.2 Segurança

Os servidores virtuais estão sujeitos aos mesmos ataques que atingem os servidores físicos, assim como novas ameaças estão explorando cada vez mais falhas no *hypervisor*. Como este gerencia todas as VMs de uma máquina física, uma vez comprometido todas as VMs também estarão. Portanto, segundo [13], a segurança total do sistema é baseada tanto na segurança do *hypervisor* quanto na do sistema operacional convidado. Se a segurança de quaisquer elementos do sistema estiver comprometida, possivelmente todo o modelo de segurança da virtualização é falho. Ataques contra o *hypervisor* serão cada vez mais comuns entre os usuários maliciosos e, portanto, os melhores *hypervisors* são aqueles que possuem atualizações constantes contra ameaças conhecidas e emergentes.

[19]De acordo com o tipo de serviço fornecido, a computação em nuvem ainda pode ser divida em mais outros dois modelos [26]: *Plataform-as-a-Service* – PaaS e *Software-as-a-Service* – SaaS. O PaaS fornece ambientes para o desenvolvimento de softwares e permite que os clientes possam implantar e testar seus próprios aplicativos na nuvem, eliminando o gerenciamento de requisitos relacionados à infraestrutura. Enquanto isso, o SaaS se refere à disponibilidade de aplicativos através da Internet, porém os usuários não estão autorizados a modificar o ambiente da aplicação.

8. CONSIDERAÇÕES FINAIS

Este trabalho apresentou os conceitos básicos de virtualização de sistemas computacionais, desde as suas origens até as soluções modernas, incluindo a contribuição da virtualização na computação em nuvem e na computação verde. É importante ressaltar que a virtualização é uma ferramenta já indispensável ao bom funcionamento dos sistemas modernos, que vem sendo adaptada às novas necessidades, associadas também ao uso mais racional dos recursos computacionais. Portanto, o desenvolvimento e a utilização de sistemas computacionais virtualizados não é uma ideia recente e, além disso, vem ganhando espaço no mercado devido a melhora de desempenho dos recursos computacionais, especialmente nos *datacenters*.

9. REFERÊNCIAS

[1] M. Arnold, S. Fink, D. Grove, M. Hind, and P. Sweeney. A survey of adaptive optimization in virtual machines. *Proceedings of the IEEE*, 93(2):449 –466, feb. 2005.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.

[3] A. Baruchi and R. L. Piantola. Análise quantitativa de técnicas de virtualização como ambiente de testes. In *IV Encontro Brasileiro de Teste de Software (IV EBTS'10)*, 2010.

[4] A. Beloglazov and R. Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 577–578, Washington, DC, USA, 2010. IEEE Computer Society.

[5] A. Beloglazov and R. Buyya. Energy efficient resource management in virtualized cloud data centers. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 826 –831, may 2010.

[6] A. Carissimi. Virtualização: da teoria a soluções. In *Minicursos do Simpósio Brasileiro de Redes de Computadores (SBRC'08)*, pages 173–207, 2008.

[7] M. Curtin. Write once, run anywhere: Why it matters. *Technical Article*, 1998.

[8] E. Kohlbrenner, D. Morris, and B. Morris. The History of Virtual Machines. http://www. cs. gmu. edu/cne/itcore/virtualmachine/history.htm. Acesso em 5 jun 2012.

[9] M. A. P. Laureano. *Máquinas Virtuais e Emuladores: Conceitos, Técnicas e Aplicações*. Novatec, 2006.

[10] M. A. P. Laureano and C. A. Maziero. Virtualização: Conceitos e aplicações em segurança. In *VIII Simpósio Brasileiro em Segurança da Informação e de Sistemas*, pages 139–187, 2008.

[11] J. LeVasseur, V. Uhlig, J. Stoess, and S. Götz. Unmodified device driver reuse and improved system dependability via virtual machines. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, Berkeley, CA, USA, 2004. USENIX Association.

[12] Q. Li, Q. Hao, L. Xiao, and Z. Li. Adaptive management of virtualized resources in cloud computing using feedback control. In *1st International Conference on Information Science and Engineering (ICISE 2009)*, pages 99 –102, 26-28 2009.

[13] Y. Li, W. Li, and C. Jiang. A survey of virtual machine system: Current technology and future trends. In *2010 Third International Symposium on Electronic Commerce and Security (ISECS'10)*, pages 332 –336, july 2010.

[14] C. Maziero. Sistemas Operacionais, nov 2011. http://dainf. ct. utfpr. edu.br/ maziero/lib/exe/fetch.php/so:so-cap09.pdf. Acesso em 07 jun 2012.

[15] R. McDougall and J. Anderson. Virtualization performance: perspectives and challenges ahead. *SIGOPS Oper. Syst. Rev.*, 44(4):40–56, dec 2010.

[16] S. Nanda and T. cker Chiueh. A survey on virtualization technologies. Technical report, Department of Computer Science SUNY at Stony Brook, 2005.

[17] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, jul 1974.

[18] J. S. Robin and C. E. Irvine. Analysis of the intel pentium's ability to support a secure virtual machine monitor. In *Proceedings of the 9th conference on USENIX Security Symposium - Volume 9*, SSYM'00, pages 10–10, Berkeley, CA, USA, 2000. USENIX Association.

[19] M. Rosenblum. The reincarnation of virtual machines. *Queue*, 2(5):34–40, jul 2004.

[20] M. Rosenblum and T. Garfinkel. Virtual machine monitors: current technology and future trends. *Computer*, 38(5):39 – 47, may 2005.

[21] J. Sahoo, S. Mohapatra, and R. Lath. Virtualization: A survey on concepts, taxonomy and associated security issues. In *2010 Second International Conference on Computer and Network Technology (ICCNT'10)*, pages 222 –226, april 2010.

[22] J. E. Smith and R. Nair. The architecture of virtual machines. *Computer*, 38(5):32–38, may 2005.

[23] G. Vallee, T. Naughton, C. Engelmann, H. Ong, and S. Scott. System-level virtualization for high performance computing. In *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. 16th Euromicro Conference on*, pages 636 –643, feb. 2008.

[24] A. Vasudevan, J. M. McCune, N. Qu, L. Van Doorn, and A. Perrig. Requirements for an integrity-protected hypervisor on the x86 hardware virtualized architecture. In *Proceedings of the 3rd international conference on Trust and trustworthy computing*, TRUST'10, pages 141–165, Berlin, Heidelberg, 2010. Springer-Verlag.

[25] Xen. Users' manual: Xen v3.3. http://bits. xensource. com/Xen/docs/user.pdf. Acesso em 08 jun 2012.

[26] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, pages 7–18, 2010.