UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

JOÃO PAULLO VIEIRA DE ALMEIDA

Impacto de plataformas de virtualização no consumo energético: um estudo comparativo entre Xen e KVM

Trabalho de Graduação.

Prof. Dr. Alexandre Carissimi
Orientador

Porto Alegre, junho de 2012.

Agradecimentos

Agradeço a todos que, direta ou indiretamente, contribuíram para a elaboração deste trabalho.

Sobretudo, agradeço ao professor Alexandre Carissimi, que sempre se mostrou disposto e motivado a me ajudar na orientação deste projeto e que, com seus ensinamentos, norteou o caminho até o resultado final de uma forma clara, simples e natural, como poucos sabem fazer.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ABI	Application Binary Interface
ACPI	Advanced Configuration and Power Interface
API	Application Programming Interface
ARP	Address Resolution Protocol
E/S (I/O)	Entrada/Saída (In/Out)
EPA	US Environmental Protection Agency
HAV	Hardware Assisted Virtualization
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ISA	Instruction Set Architecture
J	Joule
JVM	Java Virtual Machine
KVM	Kernel-based Virtual Machine
kWh	quilowatt-hora
MAC	Media Access Control
MMV	Monitor de Máquina Virtual
MPI	Message Passing Interface
MV	Máquina Virtual
RDRAM	Direct Rambus DRAM
SPEC	Standard Performance Evaluation Corporation
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VMCB	Virtual Machine Control Block
W	Watt

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A significativa elevação na demanda de energia elétrica nas últimas décadas – ocasionada pelo aumento da população mundial e pela adoção de tecnologias que utilizam cada vez mais os recursos energéticos disponíveis – faz com governos e empresas busquem maneiras de melhorar sua eficiência energética, evitando desperdícios e acarretando economia de recursos físicos e, por consequência, de dinheiro.

No âmbito da computação, o conceito de Computação Verde traz à tona diversas técnicas que podem ocasionar melhora na eficiência energética das máquinas físicas. Uma dessas estratégias baseia-se na virtualização, a qual faz com que o hardware (máquinas físicas, servidores, etc.) seja mais eficientemente utilizado, levando a uma economia de energia relativa à alimentação dos sistemas que, após a virtualização, poderão ser desativados.

Este trabalho apresenta uma análise de duas plataformas de virtualização distintas: Xen e KVM, comparando-as em diferentes fatores tais como desempenho, dissipação de potência e eficiência energética. Será analisado o custo da virtualização para sistemas físicos e o impacto de um sistema virtualizado (com *n* máquinas virtuais executando simultaneamente), de forma que seja possível confirmar a teoria de que a virtualização pode aumentar a eficiência energética de um sistema computacional.

Palavras-Chave: computação verde, consumo de energia, virtualização, Xen, KVM.

Impact of Virtualization Platforms on Energy Consumption: a Comparative Study between Xen and KVM

ABSTRACT

The significant increase in electricity demand in recent decades – caused by global population growth and the adoption of technologies that are increasingly using the available energy resources – makes governments and companies seeks ways to improve their energy efficiency, avoiding waste and saving physical resources and therefore money.

In the context of computing, the concept of Green Computing brings up several techniques that can improve the energy efficiency of physical machines. One such strategy is based on virtualization, which turns the hardware (physical machines, servers, etc.) to be more efficiently used, leading to energy saving due to systems' alimentation that, after virtualization, may be disabled.

This paper presents an analysis of two different virtualization platforms: Xen and KVM. A comparison of different factors was performed, such as performance, power dissipation and energy efficiency. The cost of virtualization and the impact of a virtualized system (with *n* virtual machines running simultaneously) will be analyzed; so that we can confirm the theory that virtualization can increase the energy efficiency of a computing system.

Keywords: green computing, energy consumption, virtualization, Xen, KVM.

1 INTRODUÇÃO

Com o incessante aumento da população mundial, a demanda por energia elétrica torna-se cada vez maior. Em 2009, estima-se que 84% da energia elétrica produzida mundialmente foi obtida por meio de fontes não renováveis [REN21, 2011], que acarretam prejuízos ao meio ambiente. Atualmente, em virtude dos alertas cada vez mais urgentes sobre as consequências que esse aumento na demanda pode ocasionar e dos dados relativos à emissão de gases na atmosfera, governos e empresas estão, como nunca estiveram, dando maior atenção à necessidade de aumentar sua eficiência energética.

Na área da computação, o consumo de energia sempre foi uma questão importante para fabricantes de sistemas embarcados ou de computadores portáteis. A duração das baterias e a dissipação de calor de tais sistemas são aspectos imprescindíveis para o sucesso desses produtos; por esse motivo, diversos estudiosos e desenvolvedores propuseram técnicas para reduzir o consumo energético em plataformas portáteis. Todavia, a constante demanda por melhor desempenho, aliada ao aumento do custo energético, estendeu o problema do consumo de energia aos *desktops* e aos servidores.

O conceito de *Green Computing* (Computação Verde) foi introduzido em 1992, quando a *US Environmental Protection Agency* (EPA) lançou o padrão *Energy Star* [RUTH, 2009]. Trata-se de um padrão fornecido a equipamentos eletrônicos energeticamente eficientes. Ao longo do tempo, o *Energy Star* tornou-se uma certificação respeitada e reconhecida mundialmente. Atualmente, empresas buscam atingir esse padrão para seus produtos.

De acordo com [GUNARATNE et al., 2005], um dispositivo energeticamente eficiente consome energia proporcionalmente à quantidade de trabalho que ele produz. Nesse contexto, quando equipamentos com essa característica apresentam um aumento na quantidade de potência dissipada, é observado, também, um aumento proporcional na quantidade de trabalho produzida por esse aparelho. Em sistemas computacionais, a quantidade de trabalho executada pode, superficialmente, ser representada pelo número de instruções executadas pelo processador. Dessa maneira, fabricantes buscam diminuir a relação energia por instrução executada com o objetivo de melhorar a eficiência energética do sistema.

Os sistemas computacionais apresentam baixas taxas de eficiência quando estão submetidos a uma carga pequena de tarefas. Servidores, por exemplo, raramente estão em estado de completa ociosidade (*idle)*, e dificilmente operam perto de sua capacidade máxima de operação. Ao invés disso, eles trabalham na maior parte do tempo entre 10% e 50% de seu nível de utilização máximo [BARROSO et. al., 2007]. Considerando que a relação energia consumida por trabalho produzido é bastante inferior em momentos de ociosidade do processador, a diminuição da quantidade de potência dissipada não se ajusta corretamente com a diminuição da quantidade de trabalho que o computador produz [BROWN et. al., 2010]. Nesse contexto, a virtualização surge como uma alternativa para aumentar a eficiência energética de servidores.

Basicamente, máquinas virtuais (MV) fornecem uma plataforma abstrata de execução, independente da máquina física. Desta forma, diversas máquinas virtuais podem executar em uma mesma instância física, garantindo uma maior utilização dos recursos desse sistema e liberando as máquinas físicas que tiveram suas tarefas migradas para o ambiente virtualizado (possibilitando assim a sua desativação). Essa configuração é especialmente interessante em ambientes de *data centers*, onde diversas máquinas físicas provêm diferentes serviços. Ao se executar os serviços em máquinas virtuais, é possível desacoplá-lo da instância física, permitindo-se que diversos serviços sejam executados em um mesmo servidor (quando a demanda no *data center* for pequena) e garantindo-se ainda um sistema robusto, capaz de suportar picos de ocupação dos servidores.

1.1. Objetivos do Trabalho

O estudo aqui apresentado tem como objetivo analisar a eficiência energética resultante da execução de máquinas virtuais em uma máquina física. Serão abordados dois sistemas de virtualização distintos: Xen [Xen, 2011] e KVM [KVM, 2011], que são ferramentas bastante difundidas atualmente. Dessa maneira, será possível estabelecer comparativos de desempenho e de consumo de energia entre os dois e, de forma mais abrangente, entre sistemas virtualizados versus não virtualizados. Para tal, diversos *benchmarks* serão executados em diferentes cenários de ambientes virtualizados.

1.2. Organização do Trabalho

O trabalho se divide da seguinte forma: o capítulo 2 revisa os conceitos físicos básicos como energia e potência, além de apresentar estratégias para redução no consumo de energia em nível de hardware e de software. O capítulo 3 apresenta os aspectos das máquinas virtuais, seus tipos e suas características, e explica as duas plataformas de virtualização utilizadas neste trabalho: Xen e KVM. O capítulo 4 consiste na apresentação e análise dos resultados experimentais. Por fim, o capítulo 5 apresenta as conclusões, seguido da bibliografia.

2 CONSUMO DE ENERGIA

O consumo energético dos sistemas computacionais é uma questão que está cada vez mais em voga nos últimos anos. Estima-se que a área da tecnologia da informação (TI) seja responsável por cerca de 2% das emissões de dióxido de carbono na atmosfera – semelhante aos números da indústria aeronáutica [NEWING, 2010]. O conceito de Computação Verde está, portanto, ganhando grande visibilidade.

Os artifícios empregados pela Computação Verde incluem, por exemplo, melhor utilização dos recursos de servidores (grandes consumidores de energia), otimização da dissipação do calor em máquinas físicas, equipamentos energeticamente mais eficientes, redução na quantidade de dados e de aplicações. Atualmente, os processadores estão ficando cada vez mais "verdes", visto que cada nova geração desses componentes apresenta transistores de dimensões reduzidas, que requerem menos energia para funcionar. Além disso, os fabricantes modificaram o paradigma de melhorar o desempenho aumentando a velocidade dos processadores – fato que aumenta o consumo de energia – para a utilização de múltiplos processadores (ou *cores*) no mesmo chip.

Neste capítulo, serão apresentadas algumas das técnicas de computação verde existentes. Primeiramente, será feita uma pequena introdução sobre conceitos físicos como, por exemplo, energia e potência, e como eles são utilizados em diferentes componentes computacionais. Em seguida, serão apresentadas técnicas em nível de hardware e de software que diminuem o consumo energético em plataformas computacionais. Por fim, será abordado o fato de se utilizar a virtualização como estratégia para diminuir o consumo de energia.

2.1. Conceitos Básicos

Energia é uma grandeza física que representa a capacidade de realizar trabalho. Ela é medida em Joules (J) ou em quilowatt-hora (1 kWh = 3600 kJ). Potência é a taxa com que a energia é transferida, medida em Watt (W = Joules por segundo). A relação entre energia e potência é:

$$P = \frac{dE}{dt} \qquad (equação\ 2.1)$$

Sendo *P* a potência, *E* a energia e *t* o tempo.

Em sistemas elétricos, a potência pode ser definida como o trabalho realizado pela corrente elétrica em um determinado intervalo de tempo. Em um sistema de corrente contínua (*Direct Current* – DC) no qual a tensão (V), medida em Volts, e a corrente (I), medida em Ampères, se mantém constantes durante um dado período, a potência transmitida é:

$$P = V.I \qquad (equação\ 2.2)$$

Sabendo que $V = RI$, podemos chegar à expressão:

$$P = I^2R \qquad (equação\ 2.3)$$

Sendo R a resistência do circuito, medida em Ohm (Ω).

Para o caso de sistemas de corrente alternada (*Alternating Current* – AC), a tensão é comumente expressa em V_{RMS} (*Root Mean Square*, em inglês). A corrente, por sua vez, é I_{RMS}. Em sistemas desse tipo, a corrente *I(t)* é uma função dependente do tempo – e, portanto, a potência instantânea também é – logo, a equação 2.3 precisa ser estendida para refletir esse fato. Se a função é periódica como ocorre, por exemplo, na rede elétrica pública de transmissão, a análise deve ser feita em relação à potência média dissipada ao longo do tempo. Logo:

$$P_{média} = \overline{I(t)^2.R} = R.\overline{I(t)^2} = R.{I_{RMS}}^2 \qquad (equação\ 2.4)$$

Ou seja, o valor RMS, I_{RMS}, da função *I(t)* é o valor constante que leva à mesma dissipação de potência que a dissipação de potência média ao longo do tempo da corrente *I(t)*.

Podemos, também, estender a relação para um sistema com tensão não constante, *V(t)*, com valor RMS expresso por V_{RMS}. Temos:

$$P_{média} = \frac{{V_{RMS}}^2}{R} \qquad (equação\ 2.5)$$

A potência apresentada por um dispositivo de dois terminais é o produto da diferença de potencial entre os terminais e a corrente que passa pelo dispositivo. Pode-se aplicar essa definição aos sistemas de corrente alternada, utilizando o conceito de RMS visto anteriormente. Logo:

$$P_{média} = V_{RMS}.I_{RMS} \qquad (equação\ 2.6)$$

É importante ressaltar que as equações aqui obtidas são dependentes dos sistemas serem puramente resistivos (por exemplo: chuveiros, lâmpadas incandescentes, ferros de passar roupa), o que significa que não há cargas reativas, tais como capacitores e indutores, que são capazes de não só dissipar energia, mas também de armazená-la. Como exemplo de cargas indutivas, há os motores e os transformadores. Por sua vez, cargas capacitivas podem ser representadas por lâmpadas fluorescentes e computadores.

Quando uma tensão senoidal é aplicada numa carga resistiva, a corrente circulante pela carga acompanha instantaneamente as variações de tensão. Nesse caso, corrente e tensão estão, normalmente, em fase em um dispositivo puramente resistivo alimentado por corrente alternada, como mostra a figura 2.1.

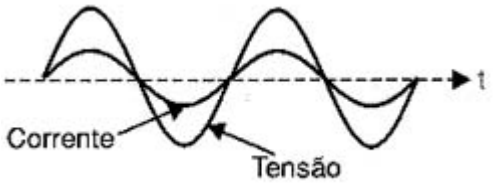


Figura 2.1: Corrente e tensão em um dispositivo puramente resistivo [BRAGA, 2010].

Contudo, na maioria dos casos reais, o sistema não se comporta como uma resistência pura. Adicionando-se componentes reativos ao sistema, o fluxo de potência flutua, ou seja, a corrente pode fluir tanto em um sentido quanto em outro ao longo do tempo. Nesse caso, a corrente não acompanha as variações de tensão instantaneamente, havendo um retardo ou um adiantamento, conforme ilustra a figura 2.2.

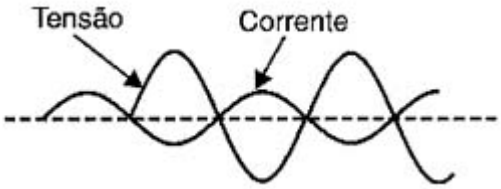


Figura 2.2: Corrente e tensão não estão em fase quando há presença de componentes reativos [BRAGA, 2010].

Nesse caso, é necessário introduzir alguns conceitos. A potência real ou potência ativa (P), medida em Watt, que representa a energia gasta em determinado espaço de tempo; a potência reativa (Q), medida em volt-ampère reativo (var), que representa a porção do fluxo de potência devido à energia armazenada que retorna à fonte a cada ciclo; a potência complexa (S), medida em volt-ampère (VA) e a potência aparente (|S|), medida em volt-ampère (VA), que é o valor absoluto da potência complexa e representa o produto da corrente pela tensão do circuito. O esquema descrito está ilustrado na Figura 2.3.

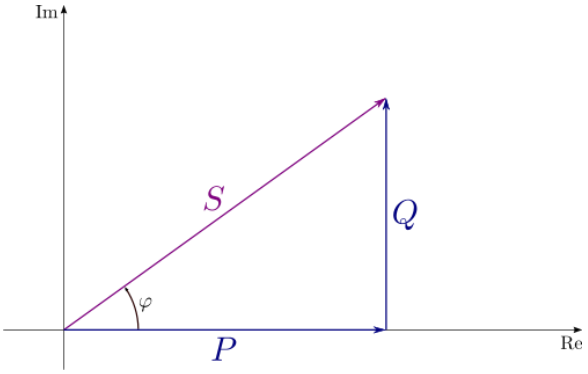


Figura 2.3: A potência complexa (S) é o vetor soma das potências real (P) e reativa (Q). A potência aparente (|S|) é a magnitude da potência complexa.

O ângulo φ formado pela potência complexa e pela potência real é usado para indicar o fator de potência (FP), que varia entre 0 e 1, e é dado por:

$$FP = cos\varphi \qquad (equação\ 2.7)$$

O melhor aproveitamento num sistema de corrente alternada ocorre quando o ângulo de defasagem é zero – ou seja, cosseno igual a 1. Nesse caso, 100% da energia aplicada é convertida em trabalho. Na prática, no entanto, equipamentos raramente atingem esse padrão. Um fator de potência baixo (φ grande) significa que energia reativa está sendo gerada e não é aproveitada [BRAGA, 2007]. Se esse desperdício ocorre em larga escala, as concessionárias de energia têm grandes prejuízos. Em virtude disso, em 1993, o extinto DNAEE (Departamento Nacional de Águas e Energia Elétrica), por meio da portaria número 1569, determinou que as instalações elétricas das unidades consumidoras devem ter um fator de potência mínimo de 0,92. Se o fator de potência estiver abaixo desse valor, a fatura mensal de energia elétrica sofre um reajuste.

Com efeito, a medida de potência em Watt de um equipamento eletrônico determina a verdadeira potência que a companhia de energia elétrica deve fornecer para o dispositivo funcionar. Por outro lado, a grandeza volt-ampère (VA) é usada para dimensionar os fios e sistemas de proteção das instalações elétricas. A classificação Watt e VA é igual para alguns tipos de aparelhos, tal como as lâmpadas incandescentes. No entanto, para equipamentos eletrônicos, essas grandezas podem se diferenciar significativamente, com a medida em VA sendo sempre igual ou maior que a medida em Watt.

Em sistemas computacionais, como visto em [CARISSIMI et al, 2010], a energia utilizada para realização de um programa executado em um processador baseados em *switches* CMOS pode ser expressa como:

$$E_{op} = C * {V_{dd}}^2 * f * \left(\frac{N}{f}\right) + I_{fuga} * V_{dd} * \left(\frac{N}{f}\right) \qquad (equação\ 2.8)$$

Onde E_{op} é a energia, C a capacitância de chaveamento, V_{dd} é a voltagem na qual o circuito está operando, f é a frequência de operação, I_{fuga} a corrente de fuga e N o número de ciclos requeridos para executar o programa.

Observando a equação 2.8, notamos que, para diminuir o consumo energético, podemos reduzir a voltagem de alimentação (V_{dd}). Contudo, por razões físicas, a frequência de processamento deve ser diminuída para que tal mudança seja possível. A redução da frequência faz com que o processador execute instruções em um ritmo menor, prejudicando o seu desempenho. Surge, então, um compromisso consumo energético versus desempenho.

2.2. Estratégias em Hardware para Redução no Consumo de Energia

De acordo com [BROWN et al, 2010], há duas técnicas básicas para gerenciamento de energia: *idle states* e *frequency scaling*. A primeira defende que o dispositivo de hardware deve ter a capacidade de desativar seus componentes que não estão em uso. Em longos períodos de ociosidade, diferentes períodos de inatividade – *idle states* – podem ser definidos. Dessa forma, o dispositivo pode desativar gradualmente as suas funcionalidades. Essa estratégia considera que, se o dispositivo está há um longo tempo sem uso, ele provavelmente ficará ainda muito mais tempo sem ser utilizado. No entanto, deve-se mensurar o custo – em termos energéticos e temporais – da reativação do dispositivo. Quanto mais profundo o *idle state*, menos energia ele consome, todavia, o custo da reativação do componente é maior. Logo, o dispositivo deve permanecer sem uso tempo suficiente para que tais custos sejam compensados. Processadores, memórias e discos utilizam *idle state* para reduzir consumo energético.

A segunda técnica – *frequency scaling* – busca a redução do consumo à custa do desempenho. Como visto na equação 2.8, a diminuição da frequência de processamento reduz linearmente seu consumo de energia e permite, também, a redução da tensão de alimentação do processador. Dessa maneira, reduzir a frequência de processamento provoca um importante impacto da potência dissipada. Processadores modernos apresentam ferramentas de hardware que permitem variar a frequência de processamento. Essa variação pode ser feita sob demanda ou conforme alguma política definida pelo sistema operacional ou, ainda, pelo usuário.

Nesta seção, serão apresentadas funcionalidades e ideias que visam melhorar a eficiência energética de dispositivos de hardware tais quais processadores, memórias e dispositivos de E/S.

2.2.1. Reduzindo o Consumo Energético do Processador

Em virtude da grande quantidade de potência dissipada pelos processadores, diversas soluções foram introduzidas em CPUs modernas para melhorar sua eficiência energética.

ACPI (*Advanced Configuration and Power Interface*): Lançada em dezembro de 1996, a ACPI especifica interfaces e conceitos para gerenciamento de potência, integrando sistema operacional, drivers, hardware e aplicação. A ACPI dá ao sistema operacional toda a responsabilidade do gerenciamento de consumo, tendo em vista que é ele que está na melhor posição para avaliar o estado de funcionamento global do sistema. A ACPI utiliza duas modalidades básicas para gerenciamento de consumo. A primeira forma é denominada *running x suspended*. Esta é a implementação dos *idle states* mencionados anteriormente. Nessa modalidade, o dispositivo pode ser desativado ou executar com menor desempenho. Para tal, a ACPI define *power states*.

Existem 7 *power states* para o sistema: C0 a C6. O estado C0 indica o estado normal de execução. Os demais estados permitem diferentes níveis de redução de consumo (*suspended states*). Na CPU, a execução de instruções pode ser suspensa e o circuito de relógio desligado. Em estados que implementam um gerenciamento de consumo mais agressivo (C5, C6), algumas partes do hardware são desativadas (caches, *buffers* e controladores de memória). Quanto mais partes do dispositivo são desativadas, mais energia é economizada. Contudo, é necessário ter em mente que estados de desativação mais profundos requerem mais tempo, e mais energia, para voltarem ao estado normal de execução. Cabe ao sistema operacional avaliar o nível de atividade do processador e fazer a troca de estado. Para permitir um gerenciamento energético eficiente, o hardware deve informar ao sistema operacional sobre o consumo para cada um dos estados, o tempo de transição entre os eles e a quantidade de energia gasta nas trocas de estado.

A segunda forma utilizada pela ACPI para gerenciar o consumo energético é baseada na ideia de *frequency scalling*, também já mencionada. São definidos *P states* (P0 a P15). A cada estado está associada uma frequência de processamento. O estado P0 é o estado de maior desempenho do dispositivo, enquanto os demais estados vão progressivamente reduzindo a frequência e a tensão de alimentação, garantindo menor dissipação de potência. O ajuste de desempenho é transparente ao usuário, pois ele não interrompe a execução do processador. Pode-se definir uma frequência de processamento constante ou variá-la conforme alguma política determinada pelo sistema operacional ou pelo usuário. Na seção 2.3.1, o gerenciamento da frequência de processamento será detalhado. Processadores modernos apresentam a possibilidade de *frequency scaling*. SpeedStep da Intel e Power Now! da AMD são exemplos de tecnologias que possibilitam o uso desse conceito.

Arquitetura Multicore: Há alguns anos, a evolução dos processadores era fácil de ser observada: bastava analisar sua frequência de operação, e compará-la com a da geração anterior, verificando um aumento na capacidade de processamento. Esse paradigma mostrou-se insustentável a partir do momento em que as altas frequências dissipavam uma quantidade tão intensa de calor que causava detrimento do próprio desempenho, além de acarretar um alto consumo energético provenientes do sistema de resfriamento. Em virtude disso, projetistas deram início à nova era: os processadores *multicore*, que utiliza diversos núcleos mais lentos em um único chip.

Um processador de dois núcleos com frequência de 1,8GHz, por exemplo, não equivale a um processador de um núcleo funcionando com frequência de 3,6GHz. Nesse caso, o trabalho é realizado por dois núcleos de 0,9GHz. Núcleos mais lentos tendem a ser mais eficientes devido a três razões [SULEMAN, 2011]. Primeiro, conforme visto na seção 2.1, a

economia de energia advém justamente da redução da frequência de processamento (proposta das arquiteturas *multicore*), que nos leva a diminuição da voltagem de alimentação (V_{dd}) i.e. economia de energia. Segundo, menos especulação. Núcleos rápidos tendem a ser mais especulativos porque eles nunca perdem tempo esperando por dados: o valor é previsto na expectativa de estar correto. Entretanto, previsões incorretas ocasionam trabalho – e tempo – perdidos. Por fim, diminuição da potência dos *flip-flops*. Núcleos rápidos tendem a possuir *pipelines* mais profundos na busca por alto desempenho. Quanto mais estágios o *pipeline* possui, maior o número de *flip-flops* no núcleo, o que nos leva a uma quantidade maior de energia desperdiçada.

2.2.2. Reduzindo o Consumo Energético da Memória

O consumo de potência em dispositivos de memórias tem crescido muito nos últimos anos. Em alguns sistemas portáteis, as memórias podem ser responsáveis por mais de 50% do consumo total do sistema [SAXE, 2010].

Baseando-se na idéia de *idle states*, fabricantes de memória estão desenvolvendo dispositivos com múltiplos estados de consumo energético. Um exemplo disso é a tecnologia Direct Rambus DRAM (RDRAM) que define quatro estados de desempenho: *active*, *standby*, *nap* e *power-down*. Para funcionamento normal de leitura e escrita, o chip deve estar no estado *active*. Os demais estados estão em ordem decrescente de consumo energético, mas em ordem crescente latência de reativação. O estado *power-down* desliga a maioria dos circuitos possíveis da memória, realizando apenas *refresh* nas células [FAN et al., 2002].

Ao utilizar essa possibilidade, o controlador de memória deve considerar o fato de que o funcionamento de um computador baseia-se na constante interação entre processador e memória. Sendo assim, um baixo desempenho da memória pode representar um gargalo no processamento.

2.2.3. Reduzindo o Consumo Energético do Disco

Comparado com o processador, os discos consomem menos energia. Um disco rígido comum de 7200 RPM consome aproximadamente 8 Watts, cerca de 10% do que um processador multicore consome [BROWN et al., 2010]. Apesar disso, a crescente demanda por espaço de armazenamento em servidores e o rápido aumento da potência dissipada em discos de última geração, tem tornado o consumo energético um problema a ser considerado nos projetos de disco rígido. Discos modernos podem ser utilizados em diferentes modos de operação. Em modos de menor consumo, pode-se limitar a utilização de caches, se reduzir o tempo de ociosidade necessário antes de se desativar o disco, etc. Algumas ferramentas existentes permitem a exploração dessas funcionalidades, conforme será visto a seguir.

Hdparm é uma ferramenta que permite parametrizar o funcionamento do disco rígido. Através dela podemos gerenciar caches, *sleep modes*, consumo energético etc. A Hdparm permite duas abordagens diferentes para o gerenciamento de consumo. Primeiro, pode-se definir 256 valores diferentes de controle de consumo. Valores baixos permitem um controle de consumo agressivo, ocasionando uma significativa redução da velocidade de rotação do disco. Valores altos, por sua vez, permitem um melhor desempenho. A segunda abordagem utilizada pela Hdparm baseia-se no tempo que o disco deve passar inativo para que o motor seja desligado. O *timeout* pode variar entre 5 segundos e várias horas. Contudo, deve-se novamente mensurar o custo temporal e energético de recolocar o dispositivo no seu funcionamento normal.

Um bom gerenciamento energético do disco requer considerações sobre seu funcionamento. Na falta de dados na cache, uma leitura obriga acesso ao disco. Uma

aplicação pode, no início de sua execução, ler todos os dados que ela vai potencialmente precisar, evitando futuros acessos e deixando o disco mais tempo em modo de menor consumo.

Laptop Mode é uma ferramenta GNU/Linux que foi criada com o intuito de manter o disco inativo por mais tempo. Considerando o custo de recolocar o disco em pleno funcionamento antes de um acesso, o sistema pode manter em cache os dados a serem escritos no disco até que o acesso ao disco seja inevitável (quando um processo faz uma leitura ou quando a cache fica completamente cheia). Deve-se, neste caso, considerar o risco de perda de dados em caso de falha no sistema ou falta de energia.

2.2.4. Reduzindo o Consumo Energético da Rede de Dados

O grande tráfego de dados da Internet consome uma enorme quantidade de energia em ambientes de rede. Os protocolos padrões nos quais as redes de computadores se baseiam, em geral, dão pouca importância à eficiência energética. Um exemplo disso é o protocolo ARP (Address Resolution Protocol). Este protocolo é usado para determinar o endereço de nível de enlace (MAC) do receptor a partir do endereço da camada de rede (IP). Ao executar o protocolo, o emissor envia por broadcast um pacote ARP para todas as máquinas da rede perguntando qual o endereço da interface de rede que possui o endereço IP informado. Desta forma, um pacote ARP vai chegar em todas as máquinas da rede e gerar uma interrupção pela placa de rede que deverá ser tratada pelo sistema operacional. Apenas o receptor (ou o default gateway, no caso da máquina alvo estar em outra rede) responderá à requisição. As demais máquinas descartarão o pacote. Esse esquema evidencia a péssima eficiência energética do protocolo.

Apesar de pouco suporte por parte dos protocolos de rede, algumas boas ideias para melhorar a eficiência energética na transmissão de dados surgiram, a maioria contando com pequenos suportes de hardware.

Wake On Lan (WOL): Em 1995 a AMD desenvolveu um método conhecido como WOL [GUNARATNE et al., 2005] para "acordar" computadores em uma rede, a partir de sua interface de rede. Em um computador em estado *sleep*, a interface de rede, ao receber um pacote WOL, vai gerar uma interrupção no computador para que este seja reativado. Esse esquema, contudo, exige o conhecimento do endereço MAC da interface de rede. Como já explicado, o emissor deve utilizar o protocolo ARP para obter tal informação. Desta forma, ao tentar acordar uma única máquina, o emissor vai acabar acordando todas as máquinas da rede, ao enviar um ARP em broadcast. Para que o esquema de WOL seja realmente eficiente, a interface de rede deve ser mais seletiva com os pacotes que exigem reativação de todo o sistema. Além disso, ela deve ter a capacidade de responder a requisições simples (ICMP, por exemplo), sem a necessidade de reativar o computador. Neste caso, pequenas mudanças no hardware padrão de uma interface de rede (como a adição de um pequeno processador) são necessárias para que ela possa efetuar esses procedimentos básicos.

Em [GUNARATNE et al., 2005] foram analisados os pacotes recebidos durante 12 horas e 40 minutos em um computador em *idle* conectado a uma rede. Constatou-se, nessa análise, que cerca de 90% dos pacotes poderiam ser descartados (na sua maioria pacotes ARP) ou exigiam mínimo tratamento. Sendo assim, ao se utilizar uma interface de rede capaz de pré-tratar os pacotes recebidos, é possível reduzir-se a frequência de reativação do sistema pela rede em cerca de dez vezes.

Concentrar Interrupções: Placas de rede modernas, ao invés de interromper o processador a cada pacote que chega, o fazem em intervalos de tempos definidos. Desta forma, possivelmente, o processador será interrompido com menos frequência e tratará vários

pacotes a cada interrupção. Deve-se tomar cuidado, no entanto, com aplicações de tempo real devido à latência temporal introduzida por tal técnica.

Escalar a Velocidade da Conexão: A potência consumida para transportar dados cresce com a largura de banda usada. Uma conexão de 1 Gb, por exemplo, consome cerca de 3 Watts a mais que uma conexão de 100 Mb (que tipicamente consome cerca de 5W a 9W) [GUNARATNE et al., 2005]. Uma abordagem interessante (baseada na redução do desempenho visando uma melhor eficiência energética) seria adaptar a taxa de transmissão de uma conexão em função da demanda. Ainda em [GUNARATNE et al., 2005], é proposto um algoritmo que analisa o tamanho da fila de pacotes em um PC, ou switch, para determinar a demanda de pacotes e alterar a velocidade da conexão.

Protocolo PS-Poll (Power Save Poll): Em uma rede IEEE 802.11, um adaptador de rede sem fio emite constantemente ondas eletromagnéticas para se comunicar com o ponto de acesso (*access point*) ou base *wireless*. A idéia do PS-Poll é fazer com que a interface de rede emita de forma intercalada, mantendo-se inativa por intervalos de tempo. Para tal, o ponto de acesso deve ser informado sobre o tempo em que a interface ficará inativa para armazenar todos os pacotes destinados a ela durante o período e os enviar posteriormente. Novamente neste caso, deve-se ter cautela com a latência temporal na distribuição dos pacotes.

2.2.5. Reduzindo o Consumo Energético de Outros Componentes de Hardware

Os monitores de LCD são grandes consumidores de energia em computadores. Visando aumentar o tempo de carga da bateria em sistemas portáteis, sistemas operacionais podem diminuem o brilho da tela ou, até mesmo, a desligar quando o usuário para de interagir com o sistema. Além disso, a Intel introduziu uma tecnologia que monitora a distribuição de cores na tela e permite a diminuição da luz de fundo quando a maior parte da tela está escura. Outra ideia é comprimir o conteúdo do *framebuffer* (buffer que armazena um quadro inteiro antes dele ser mostrado na tela). Desta forma, a quantidade de dados a serem lidos pelo hardware é diminuída. Esta otimização pode diminuir o consumo em até 5W [GARRET, 2008]. Existem ainda diversas pequenas práticas e dicas para se reduzir o consumo energético em diferentes plataformas como, por exemplo, desabilitar Wi-Fi e Bluetooth, o que pode aumentar bastante o tempo de carga da bateria de sistemas embarcados; desabilitar os logs (syslogd) do sistema pode evitar acessos ao disco e, dessa forma, poupar energia [LESS WATT, 2011], etc.

2.3. Estratégias em Software para Redução do Consumo de Energia

Além dos elementos em hardware, pode-se reduzir o consumo de energia por meio de estratégias de software, que podem ser relativas ao sistema operacional – que faz uso das funcionalidades descritas na seção anterior –, ou ainda ao conjunto de aplicações que o usuário executa.

2.3.1. Reduzindo o Consumo Energético a partir do Sistema Operacional

O sistema operacional, por meio das instruções e das chamadas de sistema, se comunica com o processador e com os outros componentes físicos. Portanto, é ele que tem o controle sobre o hardware e detém todas as informações úteis para gerenciá-lo.

Entre os métodos utilizados para reduzir o consumo energético do sistema operacional, destacam-se o *Dynamic Tick Model*, que tem como base manter o processador ocioso por mais tempo, e o *CPU Frequency Scaling*, que tem como objetivo gerenciar a frequência do processador em detrimento do desempenho.

A primeira técnica visa se beneficiar do fato de que o processador apresenta momentos de ociosidade. Do ponto de vista do processador, um estado de inatividade profundo – no qual mais partes do hardware são desativadas – otimizará o consumo energético apenas se a CPU ficar tempo suficiente nesse estado. Logo, um escalonador que gera interrupções contínuas a cada preempção (normalmente 100 vezes por segundo) pode ser bastante inconveniente quando a CPU está em estados de baixo consumo. Ao invés de utilizar um temporizador estático, o núcleo do sistema operacional, por meio do seu escalonador, pode observar os processos em execução e disparar interrupções quando considerar mais conveniente. Idealmente, um temporizador dinâmico permite ao processador ficar quase inteiramente desativado até que um usuário interaja com o núcleo por meio de uma chamada de sistema.

A segunda técnica está presente a partir do kernel 2.6.0 do GNU/Linux. É a infraestrutura chamada *CPUfreq*. Quando processadores operam em velocidades menores de relógio, eles consomem proporcionalmente menos potência e dissipam menos calor. O gerenciamento dinâmico da velocidade do relógio por meio do *CPUfreq* fornece um controle para obrigar o sistema a consumir menos energia quando não operado na sua capacidade máxima [HOPPER, 2009]. Essa solução baseia-se no conceito de governadores. Um governador é um algoritmo que calcula a frequência que ele considera adequada para a CPU em um determinado momento. Além dos governadores, é possível selecionar, manualmente, uma frequência específica de processamento. A tabela 2.1 mostra exemplos de governadores.

Tabela 2.1: Governadores para CPU *Frequency Scaling*.

Governador	Descrição
Performance	Coloca o processador na sua frequência máxima de operação. Busca máximo desempenho.
Powersave	Coloca o processador na sua frequência mínima de operação.
Userspace	Permite ao usuário ou a um programa definir uma frequência manualmente.
Ondemand	Verifica regularmente a taxa de utilização do processador. Compara essa taxa com um valor pré-definido. Se a taxa é inferior a esse valor, configura o processador na mínima frequência de operação. Quando a taxa é superior a esse valor, coloca o processador na sua frequência máxima de operação.
Conservative	Define dois valores: *up_threshold* e *down_threshold*. Verifica regularmente a taxa de utilização. Se a taxa é inferior a *down_threshold*, coloca a CPU para executar na primeira frequência mais baixa que a atual. Analogamente, se a taxa é superior a *up_threshold*, coloca a CPU na primeira frequência mais alta que a atual. É mais custoso computacionalmente que o Ondemand.

Fonte: HOPPER, 2009.

2.3.2. Reduzindo o Consumo Energético da Aplicação

As aplicações não têm acesso direto ao hardware, sendo necessário o sistema operacional para realizar essa interação. No entanto, existem diversas técnicas e recomendações que um desenvolvedor pode seguir para tornar sua aplicação energicamente mais eficiente.

***Race to Idle*:** Baseia-se no fato de finalizar as tarefas no menor tempo possível, para retornar ao estado de ociosidade (*idle*) o mais rápido possível. Um exemplo simplificado [LESS WATT, 2011] diz que um processador típico consome 34 Watts quando executado na frequência máxima ($f_{máx}$), e 24 Watts quando executado em $f_{máx}/2$ e 1 Watt quando encontra-se no estado *idle*. Supondo que executamos uma aplicação que dura 0,25 segundos na velocidade máxima. Temos, portanto, que o consumo de energia durante 1 segundo é:

- Em $f_{máx}$: $0{,}25s * 34W + 0{,}75s * 1W = 9{,}25\ Joules$

- Em $f_{máx}/2$: $0{,}5s * 24W + 0{,}5s * 1W = 12{,}5\ Joules$

Embora o exemplo acima seja simplificado, a ideia central é o que ocorre em geral em sistemas reais: é melhor executar o mais rápido possível para que se possa ficar mais tempo em estado *idle*.

Utilizar *buffers* maiores: Dados que são transferidos entre dispositivos de entrada e saída e a memória são armazenados em regiões de memória temporária - *buffers*. Ao definir o espaço de memória ocupado pela aplicação, os desenvolvedores devem criar *buffers* suficientemente grandes para minimizar o número de acessos ao dispositivo de entrada e saída. Uma aplicação que lê e executa faixas de um DVD, por exemplo, deve armazenar em *buffer* pelo menos vinte minutos de vídeo para ter uma boa eficiência energética [LESS WATT, 2011].

Desativar dispositivos após o uso: Uma aplicação, ao terminar o uso de um dispositivo de hardware, deve informar ao sistema operacional que não necessita mais daquele dispositivo. Se isso não for feito, mesmo que a aplicação seja terminada, o sistema operacional vai manter aquele dispositivo em estado de ativação, desperdiçando energia.

2.4. Virtualização como Estratégia para Redução do Consumo de Energia

Outra maneira de se reduzir o consumo de energia é por meio da virtualização, a qual desempenha um papel essencial, por exemplo, na consolidação de servidores em *data centers*, que são locais com grande índice de consumo de energia. Estima-se que, nos Estados Unidos, estas infraestruturas são responsáveis por 1.5% do consumo energético do país [VMWARE, 2011]. Um dos fatores mais importantes para esse cenário é o mau aproveitamento dos recursos de hardware.

Tipicamente, por motivos que variam desde a heterogeneidade de clientes à segurança, os *data centers* são projetados para executar um único serviço – servidor HTTP, de e-mail e de arquivos, por exemplo – em cada máquina física. O nível de carga de uma infraestrutura de TI pode depender do dia do mês ou do horário. Um *data center* deve ser projetado de forma a garantir um bom nível de serviço o tempo todo, o que, na prática, leva a um superdimensionamento de recursos. Em uma arquitetura em que há um serviço por máquina física, os recursos de cada servidor devem ser mensurados de forma a suportar os picos de utilização. Desta forma, visto que os recursos serão plenamente utilizados apenas em momentos de pico, o hardware passará a maior parte do tempo subutilizado. Além disso, como foi comentado na introdução deste trabalho, em estado de baixa carga, a CPU apresenta baixa eficiência energética. Em outras palavras, a incapacidade da infraestrutura de acomodar dinamicamente as flutuações no seu nível de utilização gera desperdício de hardware e energia.

Uma boa solução para este problema é a utilização de *data centers* baseados em máquinas virtuais. A ideia é fornecer um serviço por máquina virtual, ao invés de um serviço por máquina física. Um servidor pode executar diversas máquinas virtuais, portanto diversos serviços. Desta forma, essa nova arquitetura define *n* serviços para cada servidor físico. Com vários serviços executando em uma mesma instância física, pode-se atingir um alto nível de utilização do seu hardware. Além disso, graças à flexibilidade de ambientes virtualizados, pode-se responder dinamicamente às variações de carga no sistema. Máquinas virtuais podem ser migradas de uma máquina física a outra sem grandes custos computacionais. Deste modo, o sistema pode ser gerenciado de forma a distribuir eficientemente os serviços entre os servidores. Pode-se concentrar todos os serviços em alguns servidores, permitindo que os outros sejam desligados.

A utilização de virtualização na consolidação de servidores garante uma utilização energética eficiente por dois fatores: (i) menos hardware é necessário, pois ele é utilizado

mais eficientemente e (ii) conforme a demanda, servidores podem ser desligados, economizando a energia da alimentação desses servidores e dos seus sistemas de refrigeração. Cada servidor virtualizado pode economizar 7000 kWh de energia e deixar de emitir quatro toneladas de dióxido de carbono por ano [VMWARE, 2011].

2.5. Considerações Finais

O mundo está vivendo uma época de desenvolvimento sustentável, que segundo o WWF (*World Wide Fund for Nature*), é o desenvolvimento capaz de suprir as necessidades da geração atual, sem comprometer a capacidade de atender as necessidades das futuras gerações. É o desenvolvimento que não esgota os recursos para o futuro.

No âmbito da computação, projetistas estão cada vez mais atentos em encontrar alternativas para utilizar mais racionalmente e eficientemente os recursos energéticos em sistemas de TI. Hoje, praticamente todo tipo de hardware conta com softwares nativos para controle inteligente de energia. Há, também, as técnicas em nível de software, que são capazes de modificar elementos do sistema como, por exemplo, a frequência de processamento para atingir uma melhor eficiência energética.

Dentro desse contexto, a virtualização é uma técnica já utilizada para se aumentar a eficiência energética em plataformas computacionais. Várias máquinas virtuais podem se concentrar em uma mesma máquina física. Além disso, elas podem ser migradas de um computador para outro. Desta forma, é possível se imaginar várias máquinas virtuais em uma única instância física, aumentando a sua taxa de utilização e desocupando outras máquinas físicas. A virtualização é o assunto do próximo capítulo, que descreverá a técnica e suas vantagens.

3 MÁQUINAS VIRTUAIS

O conceito de virtualização teve origem há mais de 40 anos, na década de 60, época na qual os *mainframes* tiveram um crescimento considerável. Esses computadores eram gigantescos e muito caros. Todavia, devido à grande demanda por uso, os *mainframes* tornaram-se indispensáveis. A velocidade atingida por essas máquinas era relativamente alta, no entanto, o aproveitamento era ineficiente, visto que o gerenciamento de processos era feito manualmente por um operador. Com isso, para melhor aproveitar os recursos de processamento dos *mainframes*, criou-se, no final dos anos 60, o conceito de tempo compartilhado (*time-sharing*), que permitia a execução de vários processos. O funcionamento baseia-se na execução, pela CPU, de cada processo durante uma fatia de tempo, se revezando entre eles. Se esse intervalo de tempo é pequeno o suficiente, um usuário tem a ilusão que todos os processos estão executando simultaneamente na máquina. Em virtude disso, múltiplos usuários poderiam executar seus processos ao mesmo tempo, compartilhando o *mainframe*.

Neste contexto, a virtualização surgiu como uma importante ferramenta. Ao prover uma camada de software que oferece um ambiente completo de execução – sistema operacional, bibliotecas e aplicações –, equivalente ao de uma máquina física, as máquinas virtuais garantiam isolamento entre os diversos usuários da máquina real. Outro problema que as máquinas virtuais solucionavam dizia respeito à portabilidade de aplicações e sistemas legados, uma vez que elas permitiam abstrair os detalhes de implementação das camadas inferiores de software e hardware. Isso era importante porque havia diversos *mainframes* no mercado, cada um com seu próprio sistema operacional e aplicações.

O domínio da IBM naquela época era indiscutível. O primeiro sistema de virtualização desenvolvido foi o CP-40. Também se destacam o CP-67 e o VM/370, o primeiro computador comercial inteiramente projetado para virtualização [BUENO, 2009]. O conceito proposto pela IBM baseava-se na execução diretamente sobre o hardware de um componente chamado monitor de máquina virtual (MMV). Múltiplas máquinas virtuais podiam ser criadas pelo MMV, com cada instância rodando seu próprio sistema operacional. Atualmente, muitas máquinas virtuais baseiam-se nesse modelo.

Na década de 80, o uso de máquinas virtuais perdeu importância, já que houve uma popularização dos computadores pessoais e uma uniformização de seus hardwares. Com isso, os sistemas operacionais se restringiram a algumas poucas famílias (Macintosh, Windows e Unix), cada uma com seu público alvo e aplicações.

Contudo, em virtude do aumento do poder computacional dos processadores atuais e a popularização dos sistemas distribuídos, a virtualização ressurgiu como uma ferramenta para explorar mais eficientemente os recursos computacionais. Hoje, entre os grandes interesses dessa tecnologia estão: segurança, confiabilidade, custo, encapsulamento de um estado de computação, tolerância a falhas, criação de um ambiente seguro para testes, possibilidade de limitação de recursos para QoS, possibilidade de se executar múltiplos sistemas operacionais,

consolidação de servidores, suporte a aplicações legadas, redução do custo de gerenciamento e de energia, etc. É justamente na questão de gerenciamento e eficiência energética que este trabalho se interessa. Nas próximas seções serão discutidos os principais conceitos e aspectos envolvidos na virtualização.

3.1. Conceitos Básicos

Os primeiros conceitos que se devem ter em relação à virtualização são de instruções privilegiadas e não privilegiadas. Essas instruções compõe o conjunto de instruções do processador (*Instruction Set Architecture* – ISA), que é a interface que delimita a fronteira entre o hardware e o software. O sistema operacional – ou mesmo uma aplicação – (peças de software) interage com o processador (peça de hardware) por meio de instruções de máquina que são simplesmente códigos binários. O processador garante, também, que o sistema operacional interaja com outros componentes de hardware (memória e dispositivos de saída) através dessas instruções.

As instruções não privilegiadas – *user* ISA (interface 2 da figura 3.1) – são aquelas que não modificam a alocação ou o estado de recursos compartilhados por vários processos simultâneos, tais como modos e configurações do processador, registradores especiais e componentes de hardware. Por sua vez, as instruções privilegiadas – *system* ISA (interface 1 da figura 3.1) – podem alterar o estado e a alocação desses recursos. Em virtude disso, esse subconjunto de instruções só pode ser executado pelo núcleo do sistema operacional.

Em regra, os processos de usuário executam apenas em modo não privilegiado. Há casos, porém, em que processos precisam executar instruções privilegiadas como, por exemplo, para realizar E/S. A interface entre aplicações de usuário e hardware é provida por intermédio do sistema operacional pelas chamadas de sistema (interface 3 da figura 3.1). A aplicação gera uma chamada de sistema para o sistema operacional, e este acessa o hardware de forma adequada. Dessa maneira, uma aplicação de usuário interage com o hardware por meio do sistema operacional (chamadas de sistema) e do subconjunto de instruções não privilegiadas. A junção dessas duas interfaces é denominada ABI (*Application Binary Interface*). A ABI define, portanto, a interface entre o processo e a plataforma onde ele está executando.

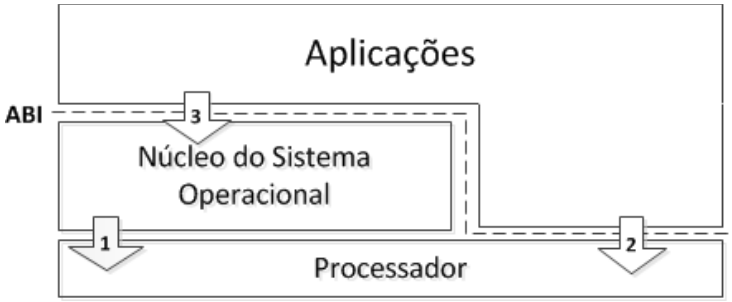


Figura 3.1: Interface entre processo de usuário e processador.

A visão de um sistema computacional como uma sobreposição de camadas de abstração que se comunicam por interfaces é a base para a construção de máquinas virtuais. Pode-se construir uma máquina virtual fornecendo uma ABI virtual a uma aplicação. Essa ABI virtual pode ser implementada por um processo de usuário (aplicação) ou por uma camada de software executando diretamente no hardware de uma plataforma ou sobre um sistema operacional já existente [CARISSIMI, 2009]. Essas duas formas de se virtualizar sistemas são a base para a construção de máquinas virtuais de processo e máquinas virtuais de sistema, que serão apresentadas a seguir.

3.2. Máquinas Virtuais de Processo x Máquinas Virtuais de Sistema

Para se compreender o conceito de máquina virtual, é importante considerar duas visões diferentes sobre o conceito de "máquina". A primeira visão é do ponto de vista de um processo executando um programa de usuário. Para ele, uma máquina consiste em um espaço de endereçamento lógico onde as instruções e dados do programa residem. A interface entre o processo e a máquina é a ABI que esta fornece. Dessa maneira, o processo pode ser executado por meio do conjunto de instruções não privilegiadas (*user ISA*) e do conjunto de chamadas de sistema oferecidas pelo sistema operacional.

Por sua vez, do ponto de vista do núcleo do sistema operacional, uma máquina é um ambiente completo de execução que pode executar múltiplos processos simultaneamente. Esses processos dividem os recursos de hardware do sistema. O sistema persiste, enquanto os processos são criados e terminados. Memória física e recursos de entrada e saída são alocados para cada processo do sistema. A interface entre o sistema operacional e a máquina é o conjunto completo de instruções da máquina (ISA).

Essas duas visões dão origem a dois modelos diferentes de máquinas virtuais: as máquinas virtuais de processo e as máquinas virtuais de sistema. As máquinas virtuais de processo executam um único processo. Elas são criadas e terminadas juntamente com o processo que executam. As máquinas virtuais de sistema, por sua vez, fornecem um ambiente persistente e completo de execução para múltiplos processos.

O processo ou sistema operacional que executa sobre uma máquina virtual é denominado hóspede. A plataforma onde a máquina virtual executa é denominada hospedeiro ou sistema nativo. A camada de virtualização que implementa a máquina virtual é chamada de *runtime* para as máquinas virtuais de processo e de monitor de máquina virtual – MMV – ou hipervisor (*hypervisor*) para as máquinas virtuais de sistema.

3.2.1. Máquinas Virtuais de Processo

Uma máquina virtual de processo oferece uma ABI virtual (*user* ISA e chamadas de sistema) a uma determinada aplicação [SMITH et al. 2005], conforme pode ser visto na figura 3.2.

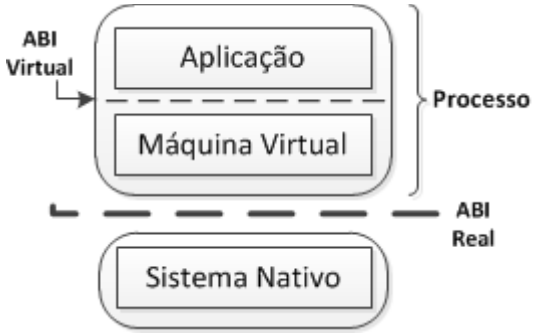


Figura 3.2: Máquina Virtual de Processo.

Um dos desafios de tal técnica é a execução de programas compilados para um conjunto de instruções diferentes daqueles do ISA da arquitetura hospedeira. A maneira mais comum de resolver esse problema é através da interpretação. Um interpretador é um programa que decodifica um determinado código, convertendo-o em um código executável. Um interpretador pode emular um código binário compilado para uma determinada arquitetura para a arquitetura hospedeira – possivelmente diferente.

Para máquinas virtuais de processo, a portabilidade é uma questão importante. No entanto, a técnica mais simples de interpretação – emulação de uma arquitetura convencional em outra – restringe a compatibilidade apenas entre as duas plataformas envolvidas. Uma forma de se atingir compatibilidade completa entre diferentes plataformas é a interpretação para um código intermediário. O código intermediário resultante pode ser visto como um código binário para uma plataforma abstrata – uma máquina virtual. O conjunto de instruções dessa máquina virtual não corresponde ao ISA de uma máquina real. A máquina abstrata é executada em uma plataforma real como uma máquina virtual de processo. Uma aplicação cujo código binário execute sobre a máquina abstrata pode executar em todas as plataformas que suportam a máquina abstrata, garantindo que um mesmo código execute em diversas arquiteturas diferentes. Um exemplo clássico desta abordagem é a linguagem de programação Java. Um código fonte escrito em Java é convertido em um código binário específico – *bytecodes* – que é executado pela JVM (*Java Virtual Machine*). O código executável pode ser portado para qualquer arquitetura que seja capaz de executar a máquina virtual Java.

3.2.2. Máquinas Virtuais de Sistema

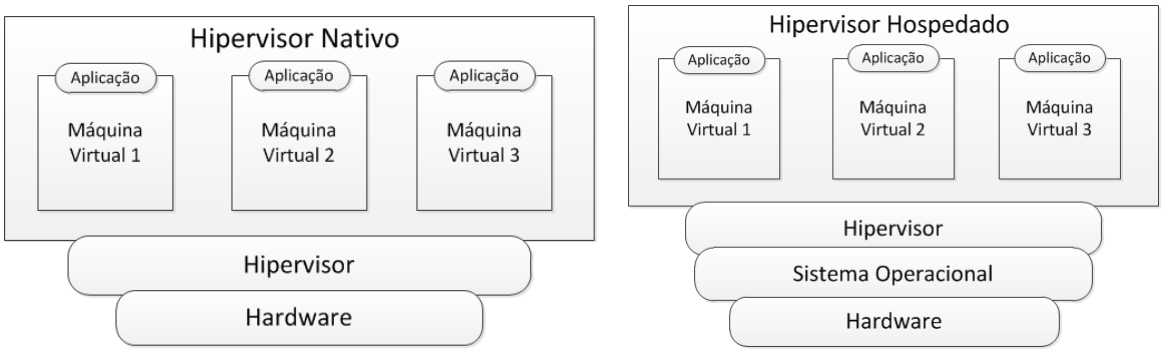
Uma máquina virtual de sistema oferece um ambiente completo no qual sistemas operacionais e processos podem coexistir. Dessa forma, uma única máquina física pode hospedar múltiplos usuários executando diferentes sistemas operacionais. Além disso, essa abordagem garante total isolamento entre processos de diferentes usuários que são executados em máquinas virtuais diferentes.

O principal desafio na implementação de máquinas virtuais de sistema é a divisão dos recursos de hardware entre os diferentes hóspedes. Cabe aos hipervisores esse controle. Existem duas formas básicas de se implementar um hipervisor de máquina virtual: hipervisores nativos (ou *bare-metal*) e hipervisores hospedados [SMITH et al. 2005].

Os hipervisores nativos executam diretamente sobre o hardware da máquina real. O hipervisor gerencia o compartilhamento dos recursos de hardware entre as diferentes máquinas virtuais fazendo com que elas tenham a ilusão de que esses recursos são privativos a elas. Esse tipo de abordagem foi inicialmente explorado pela IBM no início da década de 70. Hoje, o Xen, KVM, VMware ESX Server, são exemplos de tal técnica. A figura 3.3a ilustra um hipervisor nativo.

Os hipervisores hospedado (figura 3.3b) executam sobre um sistema operacional nativo como se fosse um processo. Neste caso, o hipervisor oferece sistemas operacionais hóspedes – possivelmente diferentes do nativo – que executa sobre um hardware virtual criado a partir dos recursos oferecidos pelo sistema operacional nativo. O VMware Player, VirtualBox e MS VirtualPC implementam essa técnica.

Pode-se identificar ainda duas estratégias diferentes na implementação de hipervisores hospedados. Uma possibilidade é oferecer uma ABI completamente diferente daquela da máquina física. A vantagem desse método é que ele permite que o sistema operacional e suas aplicações executem em plataformas diferentes das quais ela foi concebida. A desvantagem é que o hipervisor é obrigado a emular todas as instruções executadas, representando um custo no desempenho. A segunda possibilidade é oferecer um conjunto de instruções não privilegiadas idêntico ao da máquina hospedeira. Desta forma, as instruções não privilegiadas podem ser executadas diretamente, sem a necessidade de serem emuladas pelo hipervisor. A desvantagem de tal método é que as máquinas virtuais estão restritas a uma determinada arquitetura de processador.



(a) (b)

Figura 3.3: Modelos de hipervisores distintos: a) hipervisor nativo; b) hipervisor hospedado [NI, 2009].

3.3. Arquiteturas de Virtualização em Processadores x86

A arquitetura x86 possui quatro modos de operação, que representam diferentes níveis de privilégios, chamados anéis (*rings*). Os anéis são numerados de 0 a 3, onde o nível 0 representa o nível mais privilegiado na execução de instruções, sendo, portanto, o local onde o núcleo do sistema operacional executa. Por sua vez, as aplicações de usuário normalmente executam no nível menos privilegiado (*ring* 3). O esquema pode ser visualizado na figura 3.4a).

Em um ambiente virtualizado, propõe-se que o MMV execute as máquinas virtuais no *ring* 3. Assim, as instruções não privilegiadas geradas pelas máquinas virtuais podem ser executadas diretamente. Por outro lado, quando uma máquina virtual requer a execução de uma instrução privilegiada, uma exceção – *trap* – é disparada, e o MMV se encarrega do tratamento dessa instrução.

Contudo, as arquiteturas x86 possuem instruções não privilegiadas capazes de alterar os recursos do sistema, as chamadas instruções sensíveis. A arquitetura IA-32, por exemplo, apresenta dezessete instruções não privilegiadas que são consideradas sensíveis [ROBIN et al., 2000]. Para se contornar esse problema, o MMV implementado para essas arquiteturas deve tratar, além das instruções privilegiadas, as instruções não privilegiadas sensíveis. Tal alternativa ocasiona um custo em desempenho, já que as instruções *user* ISA devem ser analisadas e, as que forem consideradas sensíveis, emuladas pelo MMV. Para tal, duas técnicas são normalmente usadas. Na virtualização completa, todas as instruções sensíveis são identificadas em tempo de execução e geram um desvio para o MMV tratá-las da forma adequada. Na paravirtualização, o núcleo do sistema operacional, ao ser executado pelo MMV, é modificado para que as instruções sensíveis e instruções privilegiadas sejam substituídas por chamadas ao MMV. Essas duas estratégias de virtualização serão detalhadas nas próximas seções.

3.3.1. Virtualização Completa

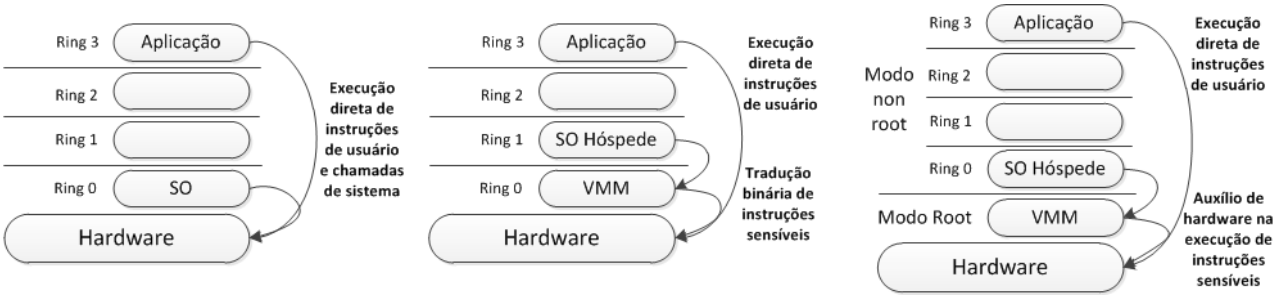
A virtualização completa (ou total) tem como ideia principal fornecer ao sistema operacional hóspede uma réplica do hardware subjacente. Para tal, o MMV deve implementar todos os componentes e os elementos relacionados ao hardware presentes na máquina física e que podem ser usados pela máquina virtual. Do ponto de vista do sistema operacional hóspede e suas aplicações, a impressão é de que eles estão executando diretamente sobre o hardware. O grande aspecto positivo dessa técnica é a transparência em relação às aplicações,

ou seja, um sistema operacional pode executar em uma máquina virtual desse tipo sem ser alterado para levar em conta instruções sensíveis e privilegiadas.

Uma desvantagem da virtualização completa é a dificuldade de simular a grande diversidade de dispositivos existentes. Para resolver esse inconveniente, um MMV fornece apenas um conjunto de dispositivos genéricos como teclado, mouse, placa gráfica e placa de rede, controladores IDE, CD-ROM, portas seriais e portas paralelas. Outro ponto negativo é o custo computacional elevado ao ter que detectar e emular as instruções sensíveis. Por fim, o último inconveniente da virtualização completa é a disputa de recursos entre os sistemas operacionais hóspedes, uma vez que cada um deles foi implementado para ser executado como uma instância única na máquina física, não havendo, portanto, controles em relação a isso. Um exemplo desse último fator negativo é o uso de paginação na memória virtual, visto que já uma disputa de recursos entre diversas instâncias de sistemas operacionais, o que acarreta em queda do desempenho.

Há duas formas de implementação da virtualização completa. A mais antiga é por meio da tradução binária [VMWARE, 2011] (figura 3.4b). Nessa abordagem, o sistema operacional hóspede opera no anel 1, e a execução de instruções sensíveis e privilegiadas causam desvio para o MMV, que as substitui em tempo de execução por uma nova sequência de instruções de forma a emular o comportamento original do código. As demais instruções não privilegiadas são executadas diretamente, sem a intervenção do MMV. Todas as instruções devem ser testadas pelo MMV para se descobrir se são sensíveis. Como as instruções sensíveis devem ser interceptadas e emuladas no hospedeiro, há um acréscimo no custo computacional da solução.

A segunda forma – e mais recente – de implementar um sistema baseado em virtualização completa é a Virtualização Assistida por Hardware (*Hardware Assisted Virtualization* – HAV). Essa solução baseia-se no suporte à virtualização por hardware fornecido pelos novos processadores (Intel VT ou AMD-V). As duas empresas desenvolveram abordagens equivalentes para o problema das instruções sensíveis. Nesses processadores, o sistema de anéis, característicos das arquiteturas x86, foi modificado para comportar a camada de virtualização.



(a) (b) (c)

Figura 3.4: Arquitetura x86 e Virtualização: a) sistema não virtualizado, b) sistema virtualizado baseado em tradução binária, c) sistema virtualizado assistido por hardware. Adaptado de [VMWARE, 2001].

Na virtualização assistida por hardware (figura 3.4c), o monitor de máquina virtual (MMV) executa em um novo modo de execução - *Root Mode* – posicionado entre o hardware e o anel 0. Além disso, algumas funcionalidades foram acrescentadas ao hardware para suportar a virtualização. Uma estrutura de dados chamada VMCB (*Virtual Machine Control Block*) registra dados de controle e de estado das máquinas virtuais presentes no sistema computacional. Uma nova instrução chamada *vmrun* é usada para se passar do modo *root*

para o modo não privilegiado – *non-root*. Ao executar *vmrun*, o hardware carrega o estado da máquina virtual a partir da VMCB. A execução em modo *non-root* prossegue até que alguma instrução privilegiada seja requisitada. Nesse momento, o MMV - usando os bits de controle do VMCB – detecta e intercepta a operação. Uma operação *exit* é gerada, passando o processador do modo *non-root* ao modo *root* e salvando o estado da máquina virtual na VMCB. O MMV, com o auxílio de informações obtidas na VMCB, trata a operação de *exit* de modo a emular o efeito da ação executada pela máquina virtual. Em seguida, o MMV executa *vmrun*, retornando ao modo *non-root.*

Notamos que ambas soluções para a virtualização completa acarretam sobrecarga ao sistema. A tradução binária, como já mencionado, pode ser onerosa devido ao custo para se analisar e emular cada instrução não privilegiada. Por sua vez, na virtualização assistida por hardware, o custo no desempenho depende da frequência que a máquina virtual irá realizar operações de *exit* e do tempo que o hardware levará para transitar entre os estados de privilégio. Uma forma de se atenuar essa sobrecarga no custo computacional seria utilizar um sistema operacional que estivesse ciente da camada de virtualização sob a qual ele está executando. Desta forma, ao se deparar com uma instrução sensível, o sistema operacional poderia chamar diretamente o MMV. Esta abordagem é a base para a paravirtualização que será apresentada na próxima seção.

3.3.2. Paravirtualização

A paravirtualização é uma alternativa à virtualização completa. Nesse modelo, o sistema operacional hóspede deve ser modificado com o objetivo de comunicar-se com o MMV sempre que houver uma instrução sensível para ser executada. Basicamente, o sistema operacional substitui todas as instruções sensíveis e instruções privilegiadas por chamadas ao MMV (*hypercall*) que, por sua vez, as interpreta e as emula da forma adequada. As instruções não privilegiadas são executadas diretamente no hardware. Dessa maneira, acaba-se com a necessidade do MMV testar instrução por instrução, o que representa um ganho de desempenho. A arquitetura está ilustrada na figura 3.5.

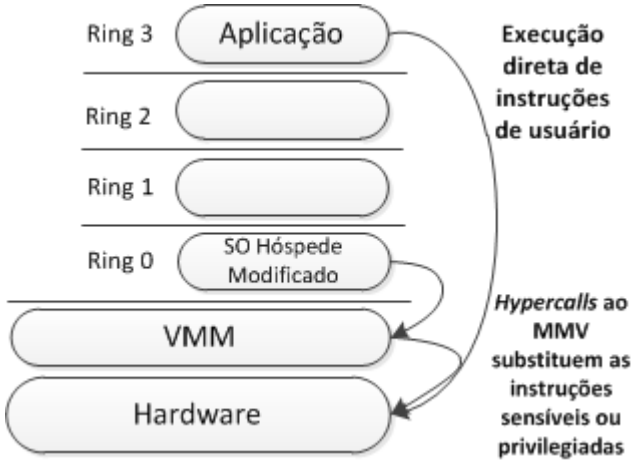


Figura 3.5: Arquitetura baseada em paravirtualização.

Outro aspecto positivo da paravirtualização é que os dispositivos de hardware são acessados por drivers do MMV, não havendo necessidade, portanto, do uso dos drivers genéricos. Com isso, ao contrário da virtualização completa, os sistemas hóspedes utilizam recursos reais da máquina. A grande desvantagem da paravirtualização é a necessidade de alterar o sistema hóspede. Essa modificação pode ser indesejada ou, até mesmo, impossível, como no caso de sistemas de código proprietário como o Windows.

3.4. Ferramentas de Virtualização Existentes

Nos últimos anos, a virtualização tem adquirido grande importância na área de TI. Aproveitando-se de toda a versatilidade dessa tecnologia, grandes corporações estão reformulando os seus ambientes TI para soluções baseadas em máquinas virtuais. A computação em nuvens, por exemplo, representa uma grande revolução na maneira com que compartilhamos dados e recursos computacionais. Esse conceito, fundamentado no uso de máquinas virtuais, é apenas um dos vários exemplos de tecnologias recentes que se aproveitam das vantagens da virtualização.

Atentos a esta revolução, diversas empresas e grupos de pesquisa desenvolveram produtos que implementam e facilitam o uso da virtualização. Nesta seção, dois exemplos de produtos existentes ilustram o mercado de virtualização atual. Serão apresentados o Xen, explicando-se a sua arquitetura particular baseada nos domínios Dom0 e DomU; e o KVM, solução disponível no próprio núcleo Linux.

3.4.1. Xen

Xen [Xen, 2011] é um monitor de máquina virtual desenvolvido em software livre, nos termos da GNU General Public License (GPL). O hipervisor Xen é compatível com várias arquiteturas, tais como, x86, x86_64, IA64, Itanium, Power PC e ARM. Ele suporta uma variedade de sistemas operacionais hóspedes como, por exemplo, GNU/Linux, FreeBSD, Solaris, Windows, etc.

O ambiente Xen consiste em vários elementos trabalhando juntos para garantir a virtualização. Podemos destacar três componentes básicos: o hipervisor, o Dom0 – um domínio privilegiado – e os DomUs – múltiplos domínios não privilegiados. O hipervisor é a camada de abstração básica que se situa entre o hardware e o sistema operacional. Ele é o responsável pelo escalonamento de processos e pelo particionamento da memória entre as diversas máquinas virtuais executando no hardware nativo. O hipervisor abstrai não só o hardware para as máquinas virtuais, mas também controla a execução das máquinas virtuais como se elas compartilhassem um ambiente comum de processamento. O hipervisor não tem qualquer conhecimento sobre a rede, dispositivos de armazenamento externos, vídeo, ou qualquer outra função de E/S encontrada em um sistema computacional; ele é apenas uma interface para as requisições desses dispositivos.

O domínio 0 (Dom0) é um sistema operacional modificado (tipicamente GNU/Linux) que atua como uma máquina virtual especial. Todos ambientes de virtualização Xen requerem que o domínio Dom0 esteja executando antes que qualquer outra máquina virtual seja iniciada. O domínio 0 funciona como um gerenciador dos domínios U, efetuando tarefas como criar, iniciar, interromper, recomeçar e destruir esses domínios. Somente ele tem direitos para acessar os recursos físicos de E/S, tornando-o responsável por tratar as requisições de E/S (acesso à rede, ao disco, etc) efetuadas pelos outros domínios hóspedes, os DomUs.

Os domínios Us, DomUs, são máquinas virtuais controladas pelo Dom0 e executam de forma independente umas das outras. Elas podem executar um sistema operacional modificado ou um sistema operacional não modificado com auxílio do hardware. Assim, o Xen oferece suporte tanto para a paravirtualização quanto para a virtualização completa assistida por hardware. A figura 3.6 ilustra a arquitetura do Xen.

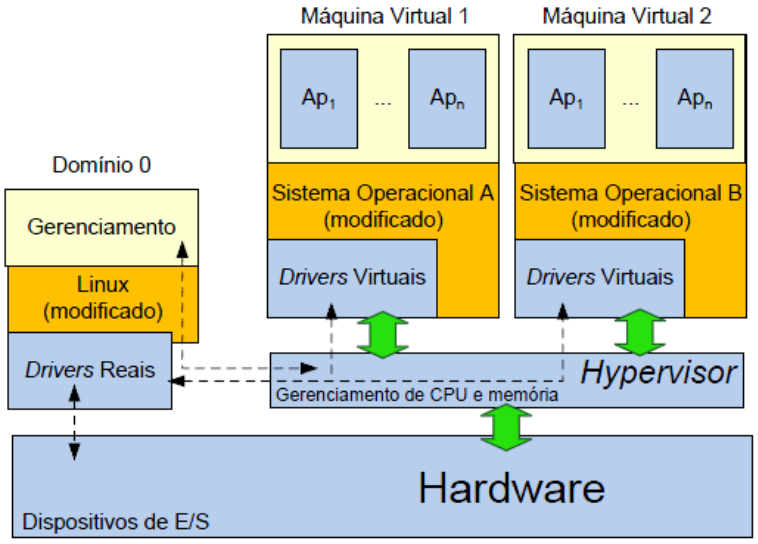


Figura 3.6: Arquitetura Xen [CARISSIMI, 2009].

Inicialmente, a escolha do Xen pela paravirtualização justificava-se pelo fato de que o ganho em desempenho era muito maior do que com a virtualização total. No entanto, com o advento das arquiteturas Intel VT e AMD-V, que dão o suporte de hardware para a virtualização, a virtualização total passou a obter resultados de desempenho melhores que os da paravirtualização, além de não haver a necessidade de modificar o sistema operacional.

3.4.2. Kernel Virtual Machine (KVM)

A abordagem realizada pelo KVM (*Kernel Virtual Machine*) é transformar o próprio núcleo Linux em um hipervisor simplesmente carregando um módulo no núcleo. Esse módulo exporta um dispositivo chamado */dev/kvm*, que ativa o modo convidado (*guest mode*, em adição aos dois modos de acesso padrão: núcleo e usuário). Com o */dev/kvm*, uma máquina virtual tem seu próprio espaço de endereçamento, separado daquele do núcleo ou de qualquer outra máquina virtual que esteja em execução. Dispositivos no diretório de dispositivos (*/dev*) são comuns a todos os processos em espaço usuário. Porém, */dev/kvm* é diferente no sentido de que cada processo que o abre enxerga um mapeamento diferente (para suportar o isolamento das máquinas virtuais). Para utilizar o KVM sem perda de desempenho, é imprescindível que o processador suporte a virtualização por hardware. Atualmente, isso significa processadores das arquiteturas Intel VT ou AMD-V.

Com o KVM instalado, é possível iniciar sistemas operacionais hóspedes em espaço usuário. Cada sistema operacional hóspede é tratado como um processo comum do sistema operacional hospedeiro (ou hipervisor) e possui seu próprio espaço de endereçamento, que é mapeado quando o sistema operacional hóspede é instanciado. A memória física que é mapeada para ele é, na verdade, memória virtual mapeada para o processo.

Observando a figura 3.7, na base, encontra-se uma plataforma de hardware que tem a extensão de virtualização por hardware. Logo acima da camada de hardware, está o hipervisor (o núcleo do Linux com o módulo KVM). Esse hipervisor se parece como um núcleo normal do Linux no qual podemos executar qualquer outra aplicação. No entanto, ele pode também suportar sistemas operacionais hóspedes, carregados por meio do *kvm utility*.

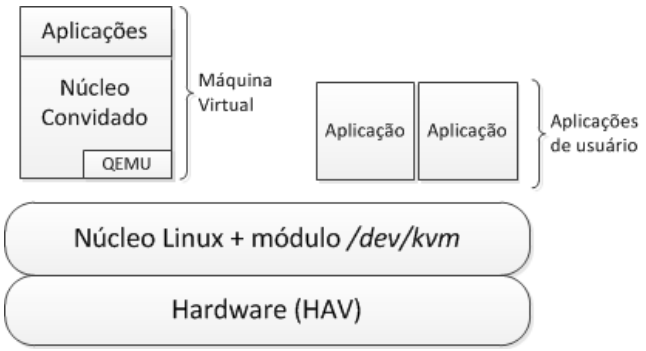


Figura 3.7: Arquitetura KVM. Adaptado de [JONES, 2007].

O tratamento de E/S de um sistema operacional hóspede é realizada por meio do QEMU [QEMU, 2012], que é uma solução de virtualização e emulação que permite a virtualização de um ambiente inteiro de um PC, incluindo discos, adaptadores gráficos, e dispositivos de rede. Qualquer pedido de E/S que um sistema operacional hóspede realiza são redirecionados para modo usuário para serem emulados pelo processo QEMU.

O modo convidado, como o próprio nome sugere, é usado para execução do código do sistema operacional hóspede. Vale lembrar que o modo núcleo representa o modo privilegiado para execução de código, enquanto o modo usuário representa o modo não privilegiado (para programas fora do núcleo). Os modos de execução são, portanto, definidos para diferentes propósitos baseado no que está executando e para qual o objetivo. Dentro do modo convidado, há os dois modos tradicionais, permitindo que uma aplicação executando como convidada possa utilizar, também, o modo núcleo, para execução dentro do núcleo, e o modo usuário, para o restante das aplicações.

É importante ressaltar que o KVM é uma parte de uma solução de virtualização. O processador fornece o suporte à virtualização diretamente. A memória é virtualizada por meio do kvm (*utility*). Finalmente, E/S é virtualizada através de um processo QEMU ligeiramente modificado, que emula hardwares como dispositivos de rede, discos rígidos, placas de vídeo, etc. Portanto, o KVM consiste em três componentes fundamentais: (i) processador com suporte à virtualização por hardware; (ii) sistema operacional com o módulo KVM ativado e (iii) componente em espaço usuário para emular o hardware do PC (processo QEMU modificado).

3.5. Considerações Finais

A escolha entre Xen ou KVM divide opiniões no mundo da virtualização. Ambas as abordagens têm vantagens e desvantagens. Como visto anteriormente, o Xen é um hipervisor externo; ele assume controle da máquina e divide recursos entre hóspedes. Por outro lado, KVM é parte do Linux e usa seu escalonamento de processos e gerenciamento de memória.

Para os defensores do KVM, um dos argumentos é a flexibilidade dessa escolha, já que os sistemas operacionais hóspedes estão se comunicando com um hipervisor que é integrado ao núcleo, sem a necessidade de modificar o sistema operacional hospedeiro. O sistema KVM se integra ao escalonador, à camada de E/S e todos os sistemas de arquivos do GNU/Linux.

O KVM foi projetado originalmente para suportar hospedeiros x86, e seu foco é na virtualização completa, sem modificações no sistema operacional hóspede. No entanto, conforme ele foi ganhando mais desenvolvedores e mais casos de uso interessantes, começou a ser portado para outras arquiteturas e a ganhar suporte à paravirtualização. Se um hóspede conseguir se comunicar com o hospedeiro, atividades como às de rede ou E/S podem ser mais

velozes. Além disso, modificações no núcleo do sistema operacional hospedeiro (Linux) para melhorar o escalonamento de processos e o uso do swap já foram propostas e aceitas.

Por outro lado, pode-se dizer que o Xen é um produto mais maduro, visto que está no mercado há mais tempo que o KVM (2003 vs. 2007). Existem implementações Xen de diversas empresas como Citrix, Novell, Oracle, Sun, Red Hat e Virtual Iron. Com isso, é mais fácil encontrar profissionais de TI com conhecimentos em Xen, mais fácil para esses profissionais receberem treinamento sobre o Xen, mais fácil localizar consultores e receber certificações Xen. A favor do Xen há, também, o fator de gerenciamento. Em comparação com o KVM, Xen apresenta uma gama mais ampla de software de terceiros para provisionamento, backup, segurança de armazenamento, monitoramento de desempenho, automação de processos, etc.

Finalmente, a paravirtualização oferecida pelo Xen faz as máquinas executarem mais eficientemente, já que elas podem acessar o hardware diretamente. O inconveniente para a utilização da paravirtualização é a necessidade de um sistema operacional modificado. Uma instalação Windows padrão, por exemplo, não funciona em um ambiente paravirtualizado. Nesse sentido, com as novas tecnologias de virtualização por hardware, o desempenho tende a superar o do modelo paravirtualizado. Cabe ao usuário definir seus objetivos e suas ferramentas para tal, e escolher a abordagem que melhor se adapta à sua necessidade.

No próximo capítulo, serão apresentados os testes realizados com as duas plataformas, de forma que seja possível elaborar, baseado em um ambiente real, um comparativo entre as duas ferramentas sob o aspecto da eficiência energética.

4 CONSUMO ENERGÉTICO EM MÁQUINAS VIRTUAIS

Um dos principais fatores a ser considerado nos sistemas computacionais atuais é que a quantidade de energia consumida não corresponde à quantidade de trabalho que o sistema está realizando. Como discutido no capítulo 1, a grande parte dos sistemas computacionais passa a maior parte do tempo operando em níveis baixos de carga de tarefas e sabe-se que, nessa situação, os componentes desses sistemas exibem eficiências energéticas ruins. Dessa maneira, com o objetivo de utilizar mais adequadamente a energia consumida por computadores, deve-se evitar que os processadores permaneçam muito tempo em estado ocioso ou de pouca carga. Para tal, quando não há trabalho a fazer, o dispositivo deve ser desativado (totalmente ou parcialmente) e, quando em funcionamento, é desejável que ele permaneça com alta taxa de utilização.

Neste contexto, a virtualização surge como ferramenta para se melhorar a eficiência energética de computadores. Como visto no capítulo 3, a utilização de virtualização em *data centers* permite desacoplar os serviços das máquinas físicas. Cada serviço provido pelo ambiente será executado em uma plataforma abstrata, uma máquina virtual (MV). Essa máquina virtual, por sua vez, poderá migrar entre diferentes máquinas físicas. Desta forma, várias máquinas virtuais – e, consequentemente, vários serviços – podem executar na mesma máquina física, garantindo um alto nível de ocupação do processador e liberando outros servidores, que podem ser então desligados.

O estudo aqui apresentado tem como objetivo realizar uma comparação entre dois sistemas de virtualização distintos: Xen e KVM. Para tal, diversos *benchmarks* serão utilizados, envolvendo análises de processador, disco e rede. Em um primeiro momento, serão observados dois aspectos: o tempo de execução e a energia consumida para a execução de cada *benchmark*. Em seguida, a intenção é observar a influência de *n* máquinas virtuais executando em um sistema físico.Com isso, pretende-se efetuar uma comparação entre as duas plataformas de virtualização e da própria virtualização em si em relação à eficiência energética.

4.1. Metodologia

Os *benchmarks* utilizados se dividem em dois grupos: os que exigem uma computação intensiva – CPU *Bound* e os que testam o desempenho e gerenciamento de dispositivos de entrada e saída – I/O *Bound*.

Cada medição foi repetida, em média, cinco vezes, já que os resultados, em cada cenário de teste específico, não apresentavam divergências significantes entre si (desvio padrão pequeno), tornando confiável o valor estatístico dessas cinco repetições. Nos raros casos onde o desvio padrão é um pouco maior, mais testes são adicionados ao ciclo, chegando a, no máximo, 10 repetições. Portanto, todos os valores que serão visto no decorrer deste capítulo são a média de um conjunto de repetições com validade estatística.

4.1.1. Plataforma Experimental

A plataforma de hardware utilizada para executar os testes é um *desktop* dotado de um processador Intel Core i3-540 – arquitetura 64 bits, dois núcleos de processamento, frequência máxima de operação de 3,07 GHz e 4GB de memória principal. O processador possui a funcionalidade de *Hyper Threading* (HT), que permaneceu desativada durante a execução dos testes. Por fim, o *desktop* possui fonte de alimentação Cooler Master 400 W com eficiência superior a 70%.

Em termos de plataforma de software, para o Xen, o monitor de máquina virtual Xen *Hypervisor* 4.0 foi utilizado para se criar o ambiente virtualizado. O Dom0 - domínio privilegiado - utiliza GNU/Linux com a distribuição Debian 6.0.3 squeeze, executando o kernel 2.6.32-5 modificado para fornecer suporte a virtualização. As máquinas virtuais – domU – são paravirtualizadas e utilizam GNU/Linux com a mesma distribuição do Dom0, mas com o kernel 2.6.32-5 modificado, neste caso, para se comunicar diretamente com o hipervisor.

Por sua vez, o módulo KVM foi ativado em um sistema operacional idêntico (GNU/Linux com a distribuição Debian 6.0.3 squeeze, kernel 2.6.32-5), para maior coerência dos resultados. A máquina física possui *dual-boot*, que permite a escolha do sistema operacional que será inicializado (com Xen ou KVM).

Todas as máquinas virtuais (Xen ou KVM) possuem processador virtual com um núcleo, 256MB de memória, disco de 8GB e rede virtual de 100Mbps.

4.1.2. Critérios para Análise

Os critérios que serão analisados nos *benchmarks* propostos são: (i) tempo de execução do *benchmark*; (ii) potência dissipada e (iii) eficiência energética, que foi um parâmetro de comparação criado para facilitar a análise dos dados. Ele é obtido a partir da razão de um valor de referência (valor ótimo) e o valor a ser estudado. Esse valor de referência depende do *benchmark* e será apresentado posteriormente.

4.1.3. Ambiente de Medição

Para obter a potência dissipada pela máquina física, foi utilizado um Nobreak SMS Manager III Senoidal, modelo µSM1400 BI FX, com potência de 1400VA. Entre as funcionalidades do aparelho, está o gerenciamento local ou remoto, por meio de um software próprio, que realiza funções como relatório de eventos, temperatura, tensão de entrada e de saída, potência, frequência da rede, etc. A tela principal pode ser vista na figura 4.1.

Pode-se observar que o dispositivo de gerenciamento informa a potência de saída como sendo uma porcentagem de 1400VA, ou seja, quando é mostrado 11% (caso da figura 4.1), significa que está sendo dissipado 154VA. A transformação para Watt se dá pela fórmula:

$$P = VA * FP \qquad (equação\ 4.1)$$

Onde FP é o fator de potência do nobreak (0,7, segundo o manual). Logo, esses 11% correspondem a:

$$P = 0{,}11 * 1400 * 0{,}7 = 107{,}8W$$

O software tem a capacidade de gerar *logs*, como pode ser visto na figura 4.2. É possível ajustar a taxa de amostragem com que os dados são gravados no sistema. Para este trabalho, foi escolhido 2 segundos para *benchmarks* mais rápidos (tempo de execução menor do que 60 segundos). Por sua vez, para *benchmarks* mais demorados (acima de 1 minuto), a taxa de gravação foi de 4 segundos ou mais. Sabendo o tempo de início e de término do *benchmark*

que está sendo executado, é possível calcular a potência média dissipada no intervalo de execução. Em seguida, podemos encontrar a energia consumida no mesmo intervalo, por meio da equação 2.1 apresentada no capítulo 2 e repetida aqui:

$$P = \frac{dE}{dt} \qquad (equação\ 2.1)$$

Figura 4.1: Tela principal do software de gerenciamento do nobreak.

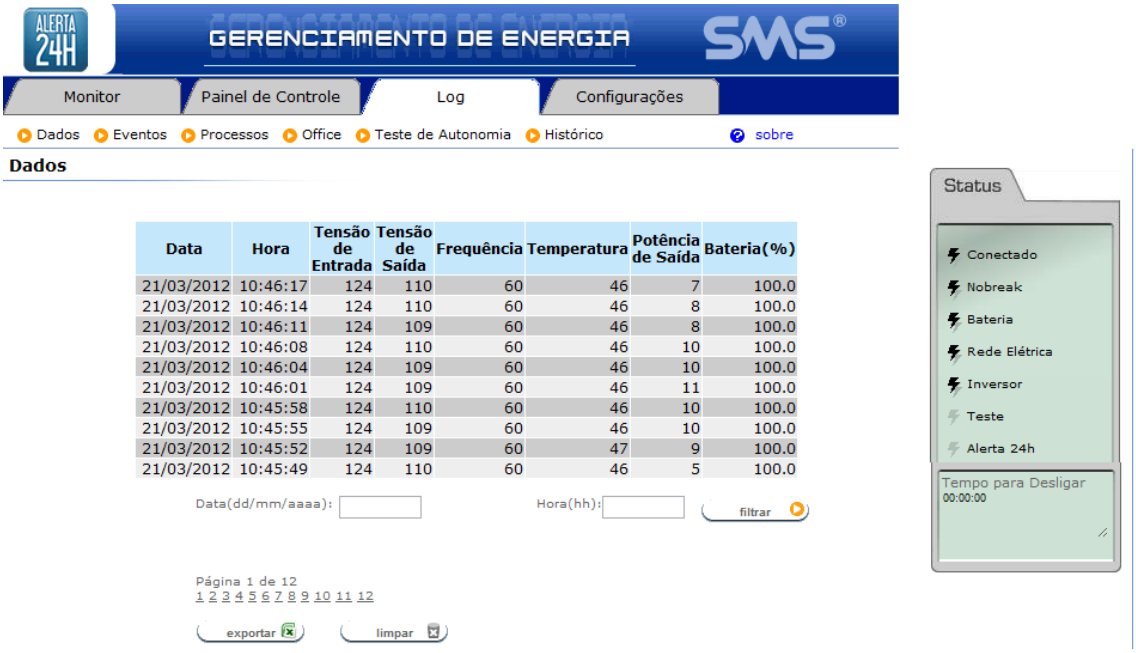


Figura 4.2: *Log* gerado pelo software de gerenciamento.

4.1.4. Benchmarks

As cargas computacionais empregadas neste estudo foram geradas pelos seguintes *benchmarks:*

SPEC CPU2006 [SPEC, 2011]: É um *benchmark* do tipo CPU intensivo desenvolvido pela SPEC (*Standard Performance Evaluation Corporation)*. O SPEC CPU2006 gera uma carga computacional que visa estressar processador e a memória através de aplicações reais de usuário. Os *benchmarks* avaliam operações aritmética com números inteiros (programas como compiladores, interpretadores, processadores de texto) e operações aritméticas em ponto flutuante (simulação física, processamento gráfico, soluções de problemas químicos complexos). Por representar uma aplicação bastante comum na computação, utilizou-se um *benchmark* que realiza compressão e descompressão de arquivos, chamado *401.bzip2*.

Iperf [IPERF, 2011]: O Iperf é um *benchmark* de entrada e saída. Seu objetivo é medir a largura de banda da conexão e o desempenho dos protocolos TCP e UDP. O Iperf permite edição de diversos parâmetros e características de transmissão. Nos testes apresentados neste trabalho, para cada plataforma de virtualização (Xen e KVM), a respectiva máquina virtual é tratada como o cliente e o Dom0 (Xen) ou o sistema operacional com o módulo KVM ativado é tratado como servidor. Tem-se, assim, uma rede virtual. O *benchmark* consistiu em transmitir 2,5 GB (valor que não torna o teste nem tão rápido, nem tão lento) de dados entre cliente e servidor. O objetivo do teste era avaliar E/S. Desta forma, por gerar menos sobrecarga de processamento, escolheu-se UDP como protocolo de transporte.

Iozone [IOZONE, 2006]: Iozone é um *benchmark* projetado para se avaliar o desempenho de sistemas de arquivos. Ele permite que diversas operações de disco sejam realizadas em diferentes configurações de funcionamento. Nos experimentos realizados, cada teste consiste em escritas e leituras de diversos arquivos com tamanho variando de forma incremental (em potência de 2), entre 32MB e 512MB. Entre outras definições, a *cache* de disco foi desabilitada, o *record size* é fixado em 64kB e o sistema de arquivos é do tipo ext3.

Com a aplicação desses *benchmarks* tentou-se avaliar o consumo energético que o hipervisor introduz em relação a um sistema nativo e o custo energético de se instanciar uma máquina virtual em cada hipervisor. Os resultados são apresentados a seguir.

4.2. Consumo Energético dos Hipervisores

O objetivo é verificar qual é o consumo energético introduzido pelo hipervisor do Xen e do KVM em relação a um sistema nativo.

As medidas de consumo energético dos diferentes *benchmarks* realizados serão utilizadas nesta seção para mensurar o impacto energético resultante da execução de máquinas virtuais em uma máquina física. Primeiramente, para o hipervisor Xen, será feita uma comparação entre o consumo energético de um *benchmark* executando no domínio privilegiado (Dom0) e em uma máquina virtual (DomU). O intuito é compreender e medir o quanto o gerenciamento e a execução das máquinas virtuais quando submetidas por uma carga computacional custa energeticamente ao sistema. Em seguida, o mesmo será feito para o hipervisor KVM: uma comparação entre a execução de *benchmarks* no sistema operacional nativo e em uma máquina virtual KVM.

4.2.1. Hipervisor Xen

Como visto no capítulo 3, o Xen implementa uma arquitetura de virtualização um pouco particular. O acesso aos dispositivos de E/S são feitos pelo Dom0, que possui os *drivers* reais dos dispositivos, sem a intervenção do hipervisor. Dessa forma, ao executar uma instrução privilegiada de E/S, o sistema operacional modificado de uma máquina virtual Xen executará uma chamada ao Dom0. Caso seja uma instrução sensível, o mesmo sistema operacional a substituirá por uma chamada ao hipervisor. Por sua vez, as instruções não sensíveis de uma máquina virtual são executadas diretamente no hardware.

O gráfico da figura 4.3 mostra o tempo de execução dos *benchmarks* quando eles são executados nos sistema nativo, no Dom0 e no interior de uma máquina virtual (DomU).

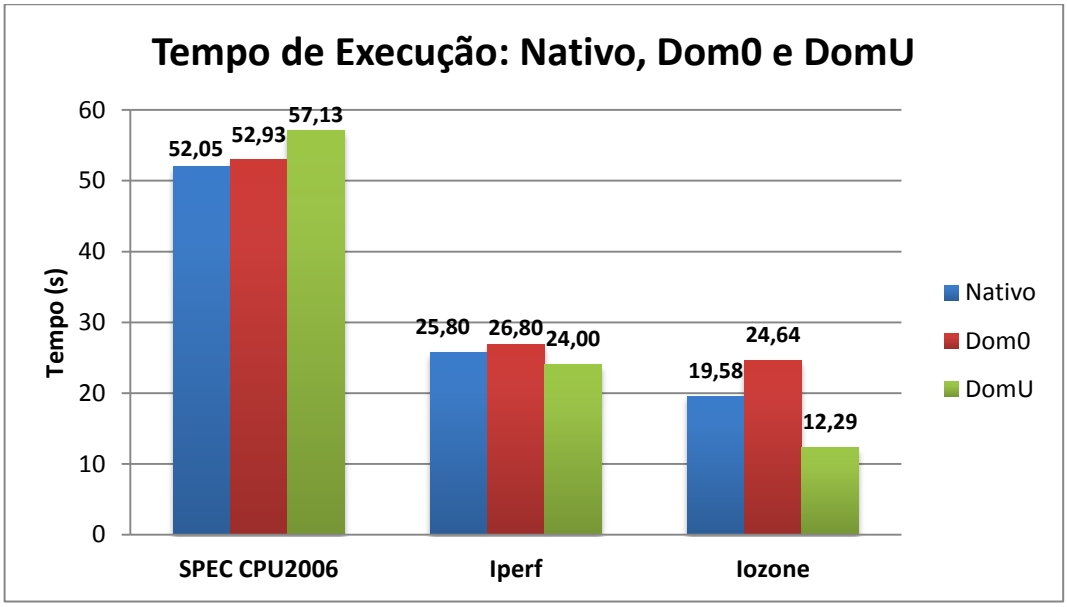


Figura 4.3: Tempo de execução dos benchmarks em sistema nativo, Dom0 e DomU.

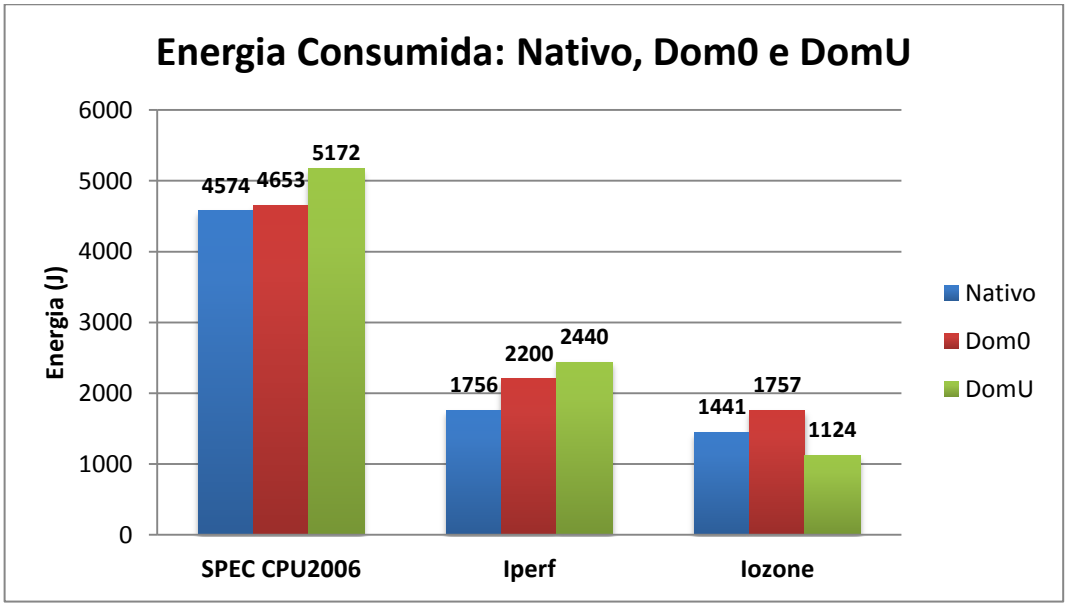


Figura 4.4: Comparação entre energia consumida pelo sistema nativo, Dom0 e DomU.

O *benchmark* CPU *Bound* (SPEC CPU2006) executa, basicamente, instruções aritméticas e lógicas (instruções não sensíveis). Em virtude disso, espera-se que o custo energético da

execução desse *benchmark* em máquinas virtuais seja similar ao custo de execução em um sistema não virtualizado. Analisando a figura 4.4, não constatamos exatamente esse comportamento. A quantidade de energia consumida pela execução dos *benchmarks* em um sistema nativo e no Dom0 varia apenas 1,7%. No entanto, em relação ao DomU, observamos um acréscimo de aproximadamente 11% no consumo energético em relação ao Dom0. Para explicar tal fato, um aspecto desse *benchmark* deve ser ressaltado: ele cria e escreve em pequenos arquivos em tempo de execução, além de, ao término da tarefa, gerar um relatório na forma de arquivo texto. Tais operações causam um sobrecusto para o domínio U, já que é necessária uma chamada ao Dom0 para tratar as requisições de E/S. Essa hipótese é confirmada com o uso do *benchmark* sintético *CPUburn*, que realiza apenas operações de cálculo sem E/S. Nesse caso, verificou-se que a energia consumida é a mesma nos três cenários (nativo, Dom0 e DomU), ou seja, o sobrecusto observado na execução do SPEC CPU2006 no DomU é devido às operações de E/S que ele realiza. Os resultados com o *benchmark CPUburn* podem ser vistos com mais detalhes em [SALGADO, 2011].

Por outro lado, os *benchmarks* I/O *Bound* (Iperf e Iozone), executam uma grande quantidade de instruções de E/S. Conforme explicado anteriormente, na arquitetura Xen, uma instrução de E/S executada por uma máquina virtual gera uma chamado ao Dom0, que é capaz de tratar a requisição, uma vez que possui os *drivers* físicos dos dispositivos. Dessa maneira, espera-se um sobrecusto energético na execução desses *benchmarks* no DomU. No Iperf, os testes ocorreram da seguinte maneira: (i) nativo: dois *shells* são iniciados: um é tratado como servidor e o outro como cliente e há a transmissão de dados entre eles conforme explicado na subseção 4.1.4; (ii) Dom0: procedimento idêntico ao executado no sistema nativo, isso é, dois *shells*, um executando o cliente e o outro o servidor e (iii) DomU: o servidor é o Dom0 e o cliente é o DomU. Nota-se que, quando efetuamos a transmissão de dados entre o Dom0 e o DomU através de uma rede virtual, é observado, na figura 4.4, um acréscimo de 39% na energia consumida em relação ao mesmo *benchmark* executando em um sistema nativo (comunicação local), embora o tempo de execução na transmissão Dom0 a DomU tenha sido menor. Esse fato pode ser explicado por meio da potência média dissipada (tabela 4.1) durante esses testes, que faz com que o valor na energia consumida aumente consideravelmente. Esse aumento da potência pode ser devido ao fato de que o processador esteja executando em estados onde há menos ociosidade ou, ainda, executando instruções que consomem mais potência.

Tabela 4.1: Tempo de execução, potência média dissipada e energia consumida nos três cenários de testes do Iperf sobre Xen.

	Tempo (s)	Pot. Média (W)	Energia (J)
Nativo	25,8	**68,08**	1756
Dom0	26,8	**82,10**	2200
DomU	24	**101,68**	2440

O Iozone apresentou um comportamento diferente do previsto: a execução em um DomU mostrou-se menos custosa energeticamente em relação ao Dom0 ou, até mesmo, ao sistema nativo. Esse comportamento se atribui, provavelmente, ao tamanho reduzido do disco da máquina virtual (8GB), que faz com que o hipervisor coloque a maior parte desse disco em memória. Sendo assim, ao executar em uma única máquina virtual, o Iozone acaba se transformando em um *benchmark* misto de disco e de acesso à memória. Como consequência, o Iozone executa muito mais rapidamente em DomU que no Dom0 (que efetivamente acessa o disco), consumindo, assim, menos energia. Quando se compara a execução do Iozone no Dom0 e no sistema nativo, novamente é constatado um sobrecusto energético de cerca de 20% oriundo provavelmente da gerência do domínio privilegiado.

4.2.2. Hipervisor KVM

No caso do KVM, não há um hipervisor independente como no Xen: o hipervisor é o próprio sistema operacional nativo com o módulo KVM ativo. Sendo assim, as medições são realizadas em máquinas virtuais KVM e no sistema operacional nativo com o módulo ligado.

O gráfico da figura 4.5 ilustra o tempo de execução dos *benchmarks*.

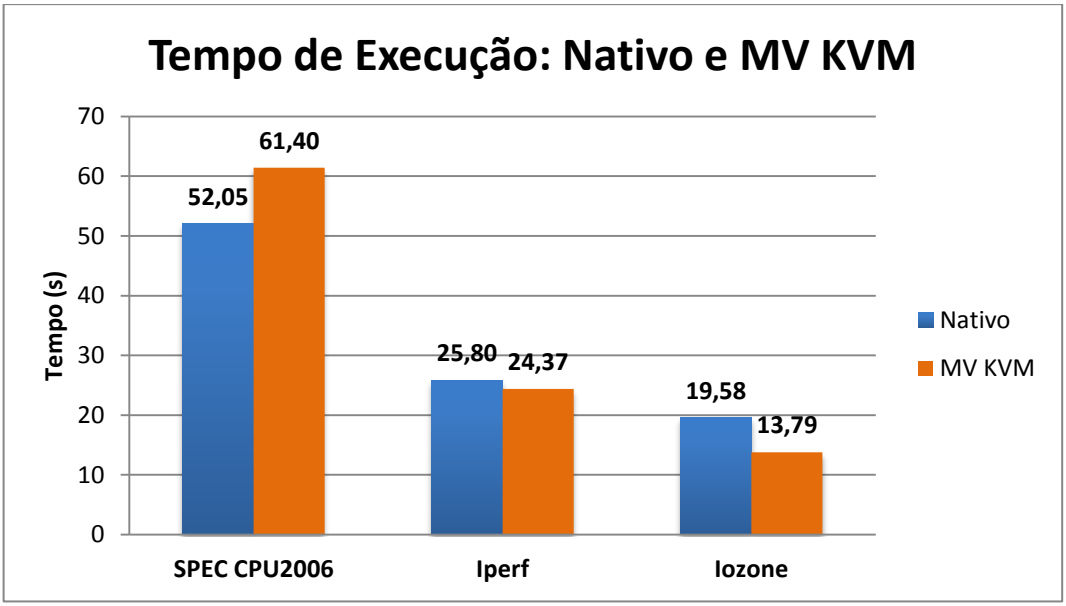


Figura 4.5: Tempo de execução dos benchmarks em sistema nativo e máquina virtual KVM.

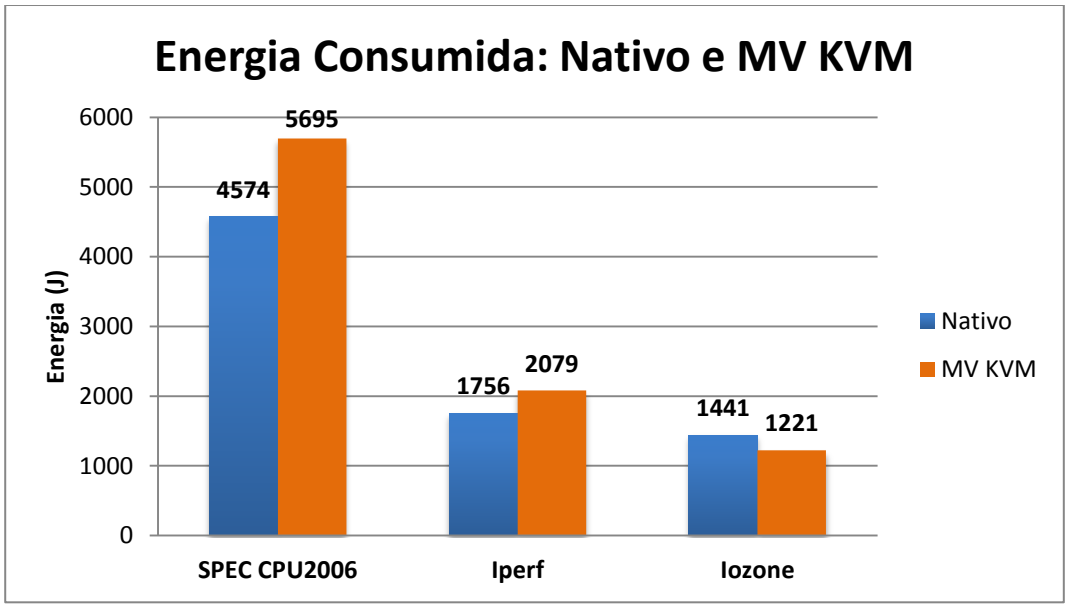


Figura 4.6: Comparação entre energia consumida pelo sistema nativo e uma máquina virtual KVM.

Analisando a figura 4.6, observa-se que, para o *benchmark* CPU *Bound* (SPEC CPU2006), o consumo de uma máquina virtual KVM foi 24,5% maior em relação ao sistema nativo. Conforme explicado anteriormente, há operações com arquivos – ou seja, instruções de E/S – durante a execução desse *benchmark*, acarretando um sobrecusto energético. Como no KVM há uma emulação dos dispositivos que realizam tarefas de E/S, esse sobrecusto

torna-se ainda mais significativo. O fato de o *benchmark* sintético *CPUburn* (100% *CPU Bound*) ter sido utilizado e não ter apresentado tais comportamentos reforça a hipótese de que o aumento no consumo de energia é devido, realmente, a tais operações de E/S. Observamos, também, que a execução do *benchmark* de compressão de arquivos em uma máquina virtual KVM demora 18% a mais em relação ao sistema nativo (61,4s contra 52,05s).

Para o Iperf, a energia consumida foi 2079 J, um aumento de 18% em relação ao sistema nativo. Esse aumento provavelmente é explicado pela emulação de dispositivos de E/S realizada pelo KVM, que pode causar um sobrecusto no sistema. Novamente, o *benchmark* em máquina virtual é executado em um intervalo de tempo menor, contudo, o consumo energético final é maior do que em relação ao sistema nativo também devido à potência dissipada, que está apresentada na tabela 4.2.

Tabela 4.2: Tempo de execução, potência dissipada e energia consumida nos três cenários de testes do Iperf sobre KVM.

	Tempo (s)	Pot. Média (W)	Energia (J)
Nativo	25,8	**68,08**	1756
MV KVM	24,37	**85,3**	2079

Para o Iozone, observamos que a energia consumida pela máquina virtual KVM é 1221 J, ou seja, 15% a menos em relação ao sistema nativo. Esse comportamento pode ser explicado pelo mesmo motivo citado para o Xen (tamanho reduzido do disco, o que faz com que o hipervisor coloque a maior parte desse disco em memória).

4.2.3. Comparação Xen x KVM

Para facilitar a análise entre os hipervisores Xen e KVM, os gráficos das figuras 4.7 e 4.8 fornecem uma comparação, lado a lado, do tempo de execução e do consumo energético deles para cada um dos experimentos realizados. Pode-se observar que, em relação ao *benchmark* SPEC CPU2006, comparando-a com o Xen, a máquina virtual KVM mostra-se mais lenta que o DomU. Esse comportamento não é estranho, uma vez que a paravirtualização (utilizada pelo Xen) é, geralmente, mais eficiente que a virtualização completa assistida por hardware (utilizada pelo KVM) para cargas intensas de acesso à memória ou operações de E/S (ou ambos). Por consequência, o hipervisor Xen (Dom0) apresentou para esse padrão de teste um consumo energético melhor que o KVM. Esses resultados vêm ao encontro daqueles apresentados em [NAKAJIMA, 2007].

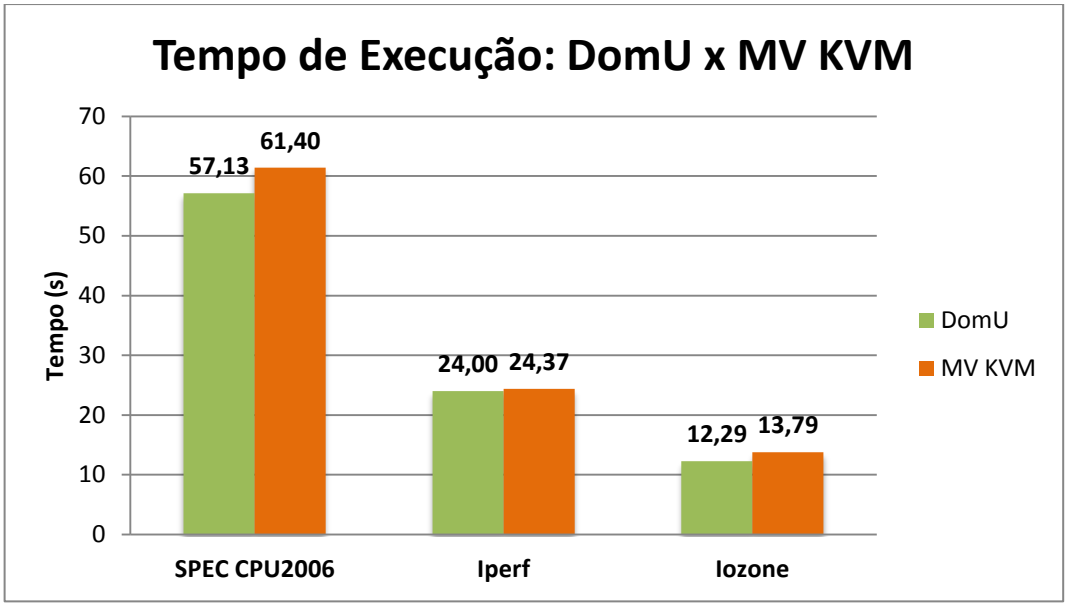


Figura 4.7: Tempo de execução (DomU x MV KVM).

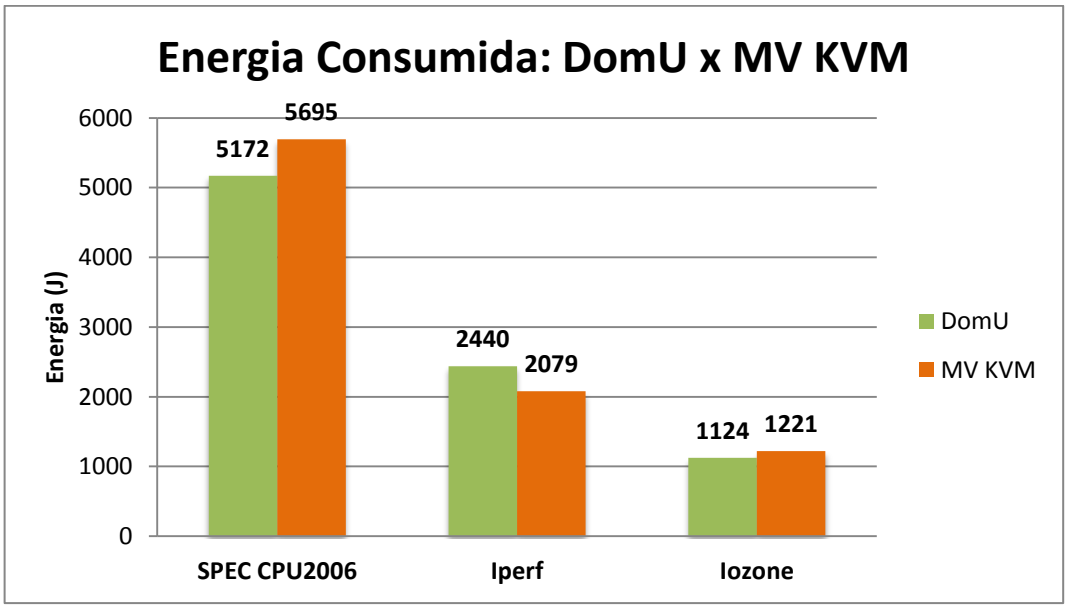


Figura 4.8: Energia Consumida (DomU x MV KVM).

No *benchmark* Iperf, o DomU consome mais energia do que o KVM, embora os tempos de execução sejam praticamente os mesmos. Isso se deve à potência dissipada. Observando a tabela 4.3, nota-se que o DomU, executando o Iperf, dissipa 19% de potência a mais do que a MV KVM (101,68W contra 85,3W), ocasionando um aumento na sua energia consumida.

Tabela 4.3: Tempo de execução, potência dissipada e energia consumida pelos três *benchmarks*.

	SPEC CPU2006			**Iperf**			**Iozone**		
	Tempo (s)	Potência (W)	Energia (J)	Tempo (s)	Potência (W)	Energia (J)	Tempo (s)	Potência (W)	Energia (J)
DomU	57,13	90,53	5172	24	**101,68**	2440	12,29	**91,47**	1124
MV KVM	61,4	92,76	5695	24,37	**85,3**	2079	13,79	**88,58**	1221

No *benchmark* Iozone pode-se notar (observando a tabela 4.3) que o DomU dissipa mais potência durante o *benchmark* em relação à MV KVM (91,47W contra 88,58W), contudo, o tempo de execução do DomU é menor, ocasionando um menor consumo de energia ao final do *benchmark*. Isso quer dizer que existem situações em que é mais vantajoso finalizar determinada tarefa mais rapidamente – onerando na potência dissipada – do que permanecer executando-a em um nível menor de potência, já que, nesse caso, o consumo energético resultante poderá ser maior. Esse comportamento é o mesmo da estratégia *Race to Idle* apresentada na seção 2.3.

4.3. Custo de Instanciar Máquinas Virtuais

Esta seção mostra o impacto energético na adição de novas máquinas virtuais ao sistema. O cenário de teste proposto consiste em efetuar a execução dos *benchmarks* SPEC CPU2006 e Iozone em *n* máquinas virtuais (*n* variando incrementalmente de 2 a 8). Com isso, pode-se analisar como a potência e o consumo energético varia à medida que mais máquinas virtuais são acrescentadas ao sistema. Como se trata de uma comparação entre duas plataformas de virtualização executando as mesmas tarefas, as subseções a seguir serão divididas pelo *benchmark* executado.

4.3.1. Benchmark SPEC CPU2006

A figura 4.9 mostra a potência dissipada pela máquina física durante a execução do *benchmark* em um DomU e em uma MV KVM.

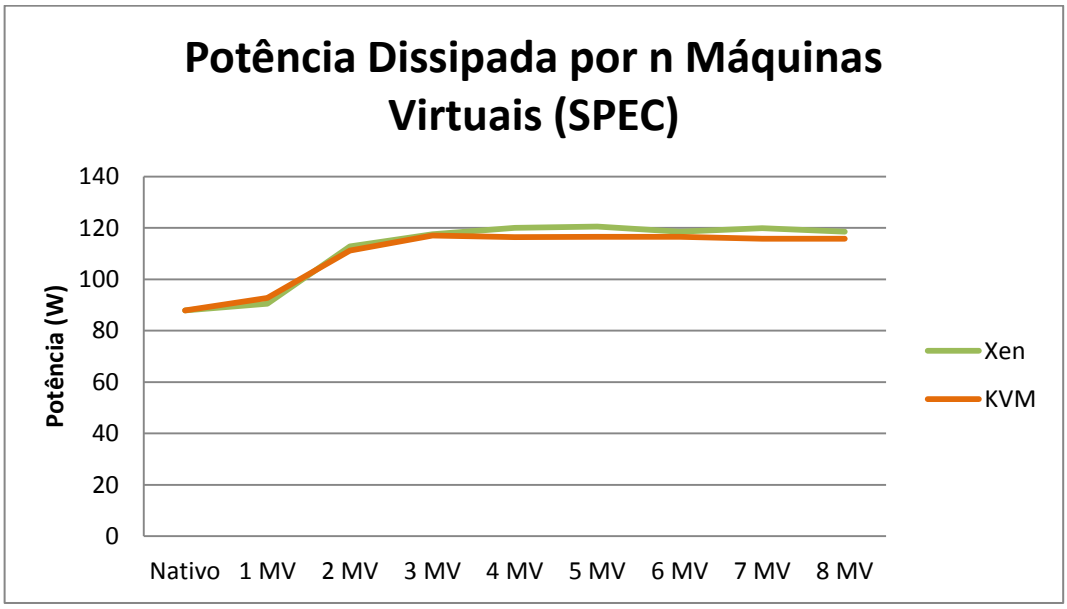


Figura 4.9: Potência dissipada por *n* máquinas virtuais (SPEC).

Para entender o comportamento da potência dissipada, deve-se considerar a arquitetura do processador e os cenários de testes. O ambiente virtualizado de testes consiste basicamente em um hipervisor e um número variado de máquinas virtuais. O processador da máquina física que hospeda esse ambiente possui dois núcleos de processamento físicos. Esses dois núcleos poderiam ser estendidos para quatro núcleos lógicos com o auxílio da tecnologia *Hyper Treading*, contudo, ela permaneceu desativada no decorrer das medições. Quando apenas uma máquina virtual está executando o *benchmark*, em média, um dos núcleos do processador será plenamente ocupado pela máquina virtual e o outro passará a maior parte do tempo ocioso, responsável apenas por eventuais tarefas de sistema (gestão de CPU e memória

pelo hipervisor, por exemplo). Ao se adicionar a segunda máquina virtual, os dois núcleos do processador serão plenamente utilizados, um para cada MV. Isso explica o grande aumento na potência dissipada pela máquina oriundo da adição da segunda máquina virtual. A partir de então, as MVs acabarão concorrendo entre si pelo processador (aumento do tempo de execução), mas a potência dissipada permanecerá praticamente constante. Observa-se, porém, um crescimento na potência dissipada entre duas e três máquinas virtuais. Isso ocorre, possivelmente, devido a trocas de contexto entre execuções de máquinas virtuais diferentes, ocasionando um sobrecusto na potência dissipada. Esse aumento atinge um valor limite, uma vez que, a partir de três máquinas virtuais, o processador tende a ficar continuamente ocupado executando chaveamentos de contexto e os códigos das próprias máquinas virtuais. Esse valor é, de fato, muito próximo da dissipação máxima de potência do processador, que é por volta de 125 W. [CPU WORLD, 2012].

Tabela 4.4: Tempo de execução do benchmark SPEC CPU2006 em *n* máquinas virtuais.

	Tempo (s)	
Sistema	**Xen**	**KVM**
Nativo	52,05	52,05
1 MV	57,13	61,40
2 MV	61,76	67,73
3 MV	87,39	97,06
4 MV	119	148
5 MV	143	197
6 MV	171	259
7 MV	206	300
8 MV	240	336

Uma vez que a potência, por si só, não é um parâmetro suficiente para fazer conclusões sobre a eficiência energética, deve-se analisar as duas variáveis restantes: o tempo de execução do *benchmark* e a energia consumida durante essa mesma execução, que é dada pela relação:

$$E = P.t \qquad (equação\ 4.1)$$

Os tempos de execução estão apresentados na tabela 4.4. Nota-se que a máquina virtual KVM tem um desempenho (em termos de tempo) inferior ao DomU. A energia consumida pode ser calculada relacionando as informações da figura 4.9 e da tabela 4.4. A figura 4.10 é o resultado dessa junção. Nela, é apresentada a energia média consumida por máquina virtual, ou seja, a energia total consumida na execução do *benchmark* dividida pelo número de máquinas que estavam executando no respectivo cenário de teste. Por exemplo, no ambiente com cinco DomUs, a energia total consumida na execução do SPEC CPU2006 foi 17234 J, que, dividido por cinco, resulta, em média, 3447 J por máquina virtual. O desvio padrão também é mostrado (em forma de barras) nesse gráfico.

O gráfico da figura 4.10 fornece informações interessantes a respeito da eficiência energética das diferentes plataformas de virtualização estudadas. Primeiramente, deve-se compreender que o resultado esperado apresentaria um menor consumo médio de energia quando há duas máquinas virtuais executando no sistema. Isso quer dizer que, nesse cenário de teste, o aproveitamento dos recursos do sistema é mais eficiente, já que há dois núcleos de processamento, e os dois estão sendo utilizados, em teoria, para executar uma máquina virtual cada. No entanto, o resultado obtido é ligeiramente diferente: para o Xen, o melhor caso ocorre quando há seis máquinas virtuais executando no ambiente de testes. Contudo,

considerando os desvios padrões, representados por barras no gráfico, pode-se dizer que, para o Xen, a partir de duas máquinas virtuais executando no sistema, a energia média consumida por cada uma delas permanece em patamares bastante próximos. Com isso, podemos montar o gráfico da figura 4.11, que ilustra a eficiência energética, considerando 100% a máquina que possui a menor energia média consumida entre as 8 máquinas utilizadas em cada um dos testes. Esse valor é a referência (há, portanto, dois valores de referência: um para o Xen e outro para o KVM). A eficiência energética é expressa pela razão do valor de referência e o valor a ser estudado. Nota-se que o Xen possui mais barras próximas a 100%, o que significa que essa forma de virtualização, para tarefas CPU *Bound*, é mais energeticamente eficiente do que a empregada pelo KVM.

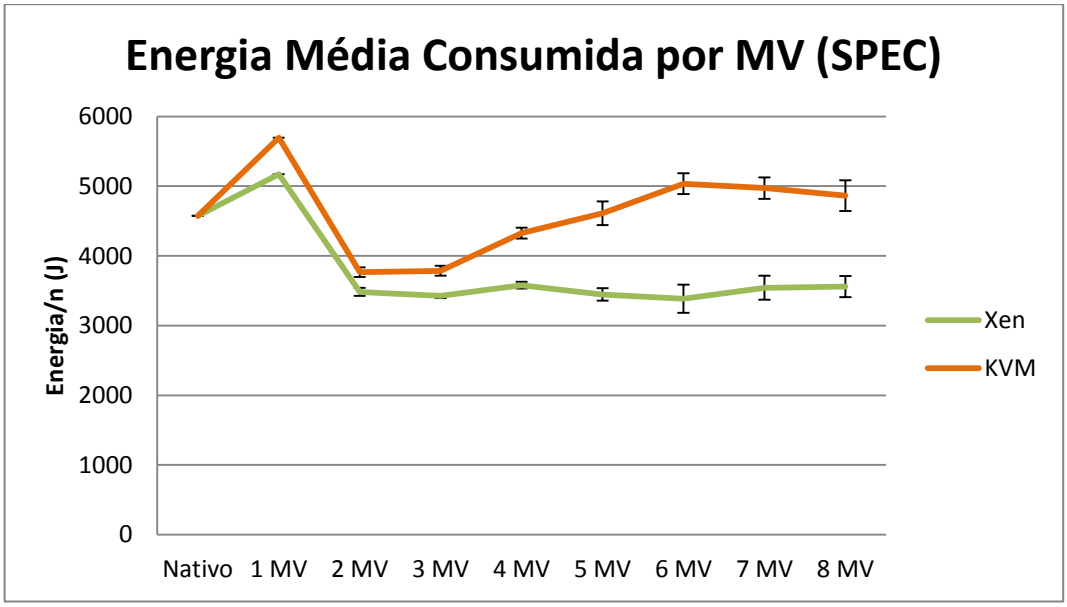
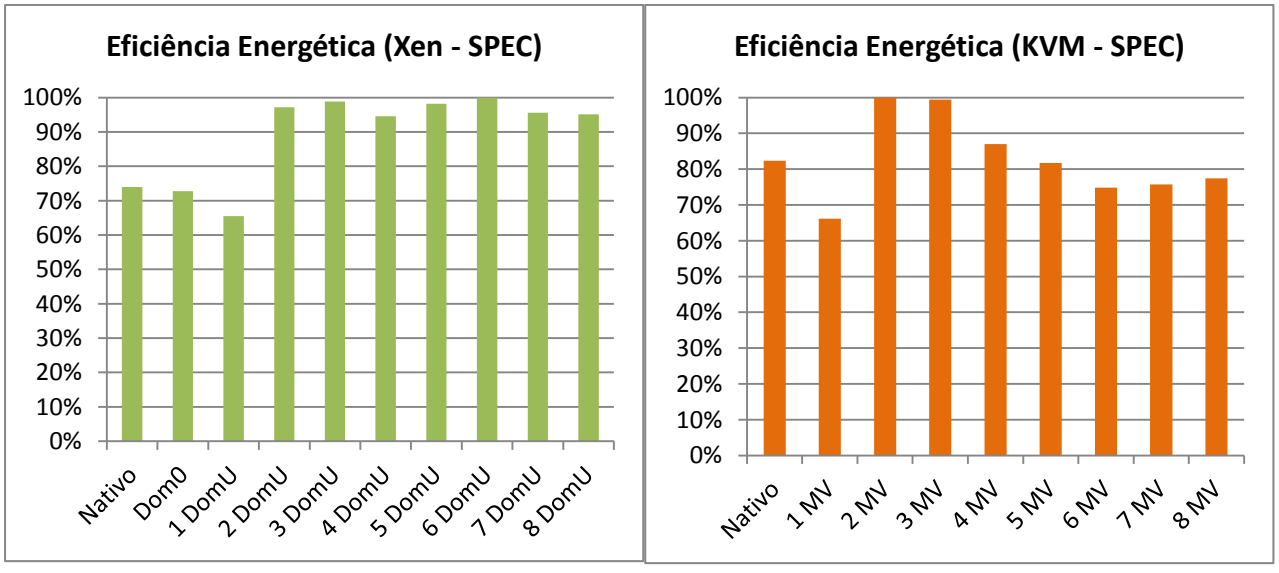


Figura 4.10: Energia Média Consumida por MV (SPEC).



(a) (b)

Figura 4.11: Eficiência Energética para o *benchmark* SPEC: a) Xen (referência é n=6); b) KVM (referência é n=2).

4.3.2. Benchmark Iozone

O *benchmark* Iozone, por se tratar de um conjunto de tarefas que realizam intensas atividades de I/O, possui um comportamento diferente do SPEC CPU2006. A figura 4.12 mostra a potência dissipada pela máquina física durante a execução do *benchmark* em um DomU e em uma MV KVM.

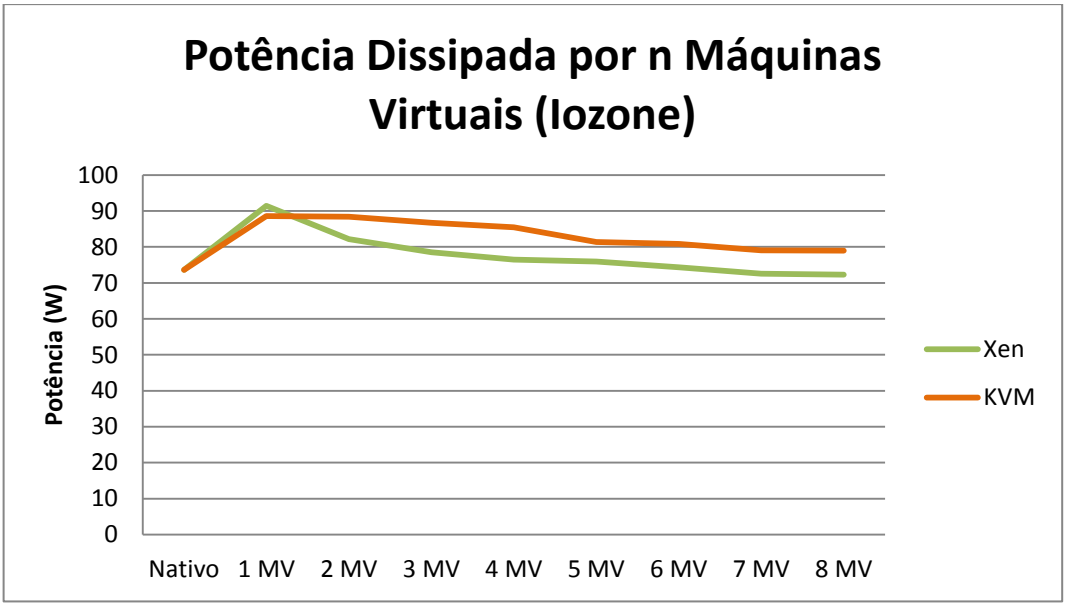


Figura 4.12: Potência dissipada por *n* máquinas virtuais (Iozone).

Analisando a figura 4.12, observa-se que também há um padrão: a potência máxima dissipada, para os dois tipos de virtualização, ocorre quando o cenário de testes está com apenas uma máquina virtual em execução. A partir de então, há uma diminuição na potência à medida que mais máquinas virtuais são adicionadas. Esse comportamento foi inesperado e uma hipótese provável é que o sistema, por algum motivo, ajuste o *power state* para um estado mais profundo de economia de energia, já que operações de E/S levam os processos a um estado bloqueado. No entanto, essa hipótese deve ser analisada mais profundamente para verificar sua veracidade ou não.

Os tempos de execução estão apresentados na tabela 4.5. Nota-se que a máquina virtual KVM tem um desempenho (em termos de tempo) superior ao DomU quando o número de máquinas virtuais no ambiente de teste está entre 2 e 6. Por outro lado, para os casos em que: (i) há apenas uma máquina virtual sendo executada ou (ii) há 7 ou mais instâncias virtuais, o DomU apresenta melhor desempenho que o KVM.

Tabela 4.5: Tempo de execução do benchmark Iozone em *n* máquinas virtuais.

	Tempo (s)	
Sistema	**Xen**	**KVM**
Nativo	19,58	19,58
1 MV	12,29	13,39
2 MV	60,33	34,33
3 MV	98,28	69,54
4 MV	132	93,18
5 MV	201	179
6 MV	247	206
7 MV	318	451
8 MV	361	641

Analisando a figura 4.13, nota-se um padrão: o menor consumo médio de energia acontece quando há apenas uma máquina virtual executando no sistema, embora, nesse caso, a potência dissipada seja máxima (conforme figura 4.12).

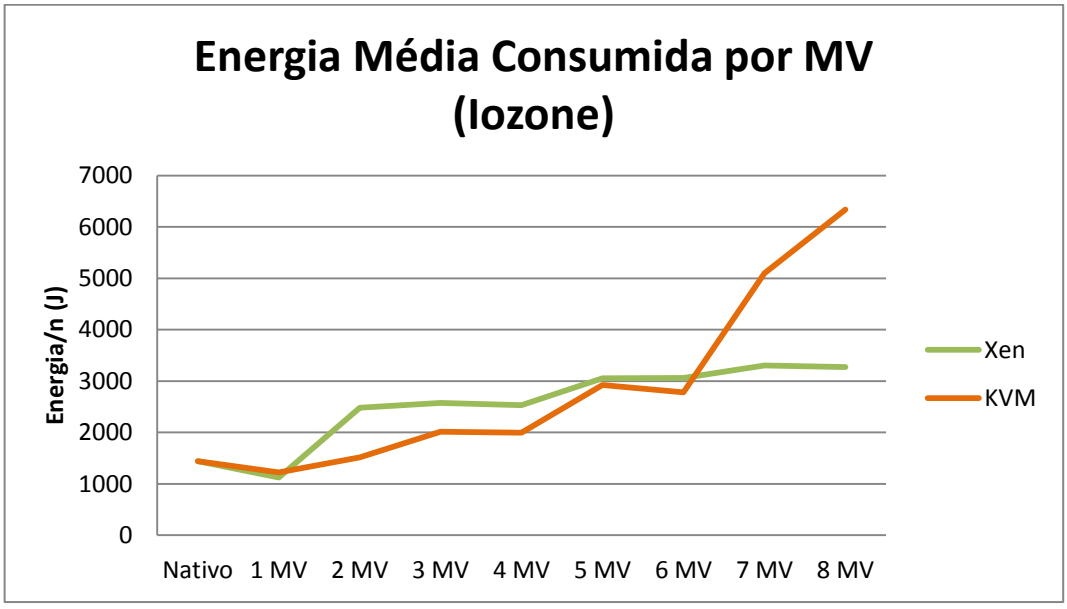
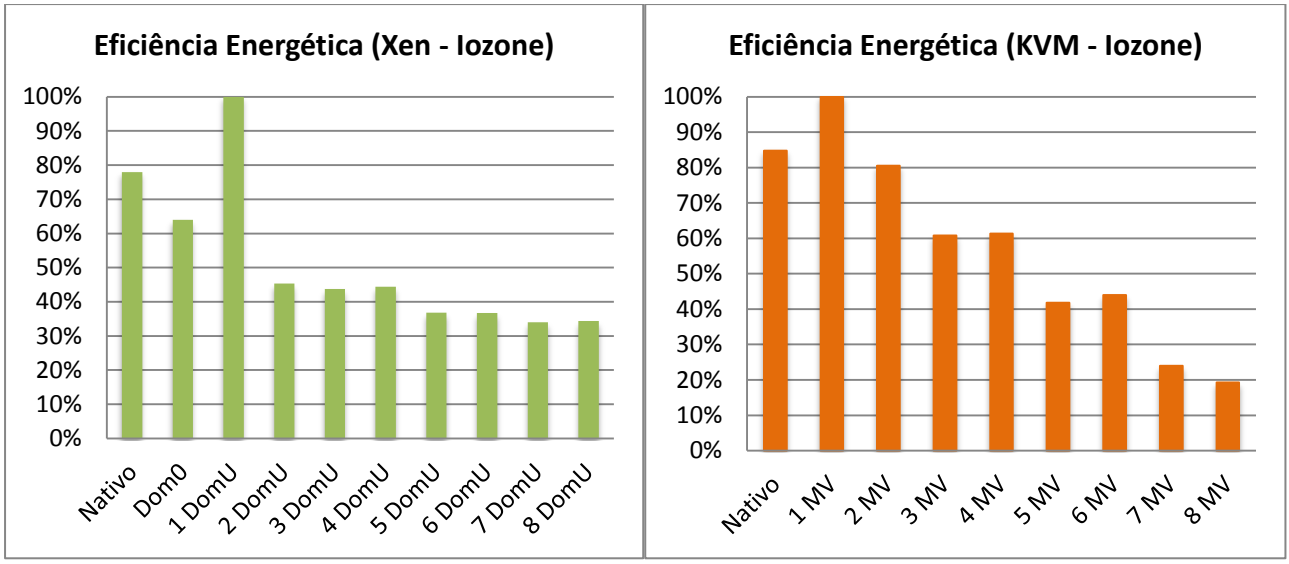


Figura 4.13: Energia Média Consumida por MV (Iozone).

O gráfico da figura 4.14 ilustra a eficiência energética, considerando 100% a máquina que possui a menor energia média consumida. Nota-se que as duas formas de virtualização apresentam máxima eficiência energética quando há apenas uma máquina virtual executando no sistema (fato que pode ser concluído analisando apenas a figura 4.13). Porém, no gráfico da figura 4.14, fica mais claro que o KVM mostra-se mais energeticamente eficiente até cenários de testes com seis máquinas virtuais executando no sistema. A partir de então, há um comportamento inesperado por parte do KVM, que tem sua energia média consumida incrementada abruptamente. Nos experimentos realizados, não foi possível identificar o porquê desse comportamento anômalo, contudo, uma hipótese é de que, como a potência dissipada nesses casos diminui gradativamente e o tempo de execução aumenta, há um provável bloqueio de processos e um gargalo de E/S no QEMU para atender as *n* MVs.



(a) (b)

Figura 4.14: Eficiência Energética para o *benchmark* Iozone: a) Xen (referência é n=1); b) KVM (referência é n=1).

4.4. Considerações Finais

A utilização de máquinas virtuais apresenta diversos benefícios, entre eles, a consolidação de servidores, que garante uma utilização mais eficiente dos recursos de hardware disponíveis, uma vez que mantém os processadores ocupados executando MVs. Isso quer dizer que, considerando um sistema computacional que consome x Joules e dissipa y Watts, a utilização de máquinas virtuais em um único ambiente torna o consumo energético final e a potência dissipada total menor do que n.x Joules e n.y Watts, respectivamente. Com isso, pode-se dizer que a utilização de máquinas virtuais garante um melhor desempenho energético da máquina física.

Nos testes apresentados, o custo da virtualização para um *benchmark* 100% *CPU Bound* – caso do *CPUBurn* – é praticamente zero. Quando é utilizado o SPEC CPU2006, que possui uma pequena porção de E/S, já nota-se um custo maior para virtualizar o sistema, principalmente quando é usado o KVM, que apresenta uma pior eficiência energética em relação ao Xen para esse tipo de *benchmark* como pode ser visto na figura 4.10.

Por sua vez, os testes com o *benchmark I/O Bound* revelaram que o grande custo da virtualização é, de fato, neste tipo de tarefa, uma vez que há um gargalo no Dom0 e no QEMU devido ao bloqueio de processos ocasionados pelas operações de E/S. Embora a eficiência energética tenha diminuído à medida que foram adicionadas mais máquinas virtuais neste cenário de teste, a virtualização ainda assim mostra-se uma solução energeticamente viável em virtude da consolidação de servidores mencionada anteriormente.

5 CONCLUSÃO

Nos últimos anos, o significativo aumento da demanda na área de TI de empresas, instituições e governos fez com que o custo para manter esses sistemas funcionando corretamente aumentasse da mesma forma. Estimou-se que, em 2007, os *data centers* eram responsáveis por 1.5% de todo consumo energético dos Estados Unidos. Com isso, dois problemas vêm à tona. Primeiro, do ponto de vista financeiro, o custo de energia elétrica para alimentar essas infraestruturas cresce a cada ano. Segundo, do ponto de vista ambiental, a poluição gerada pela produção dessa mesma energia elétrica aumenta na mesma proporção, visto que a grande parte da produção elétrica mundial é obtida por meio de fontes não renováveis.

Em virtude disso, o termo Computação Verde vem sendo cada vez mais debatido. Essa abordagem busca aumentar a eficiência energética de qualquer componente eletrônico até grandes sistemas como *data centers*, por exemplo. Para tal, diversas técnicas são propostas, tanto em nível de hardware (processador, memória, disco, rede, etc.) quanto em nível de software (funcionalidades presentes no sistema operacional e técnicas que podem ser usadas nas aplicações de usuário). Além disso, a virtualização mostra-se como uma ótima ferramenta para aumentar a eficiência energética de sistemas, uma vez que um ambiente virtualizado apresenta um alto nível de utilização de hardware, ocasionando um aproveitamento mais eficiente dos recursos de uma máquina física.

Os testes apresentados neste trabalho tiveram como objetivo compreender como as máquinas virtuais consomem energia em diversos cenários de uso. Para tal, foram analisadas duas plataformas de virtualização (Xen e KVM), de forma que foi possível efetuar uma comparação entre as duas.

A partir dos resultados apresentados no capítulo 4, constatou-se que, para o *benchmark CPU Bound*, o Xen é mais eficiente do que o KVM em termos de tempo de execução para qualquer número de máquinas virtuais executando no sistema. O Xen apresentou, ainda, consumo energético médio por máquina virtual menor do que o KVM também para qualquer número de instâncias virtuais executando no ambiente de teste. Esses resultados vêm ao encontro daqueles apresentados em [NAKAJIMA, 2007], que diz que a paravirtualização (utilizada pelo Xen) é, em geral, mais rápida que a virtualização completa assistida por hardware (utilizada pelo KVM) para cargas intensas de acesso à memória ou operações de E/S (ou ambos).

Por outro lado, para o *benchmark I/O Bound*, a execução de instruções de E/S se mostrou mais custosa em relação a instruções de acesso à memória do teste anterior. A menor energia média consumida – e, portanto, a maior eficiência energética – ocorre no cenário em que há apenas uma máquina virtual executando no sistema, tanto para o Xen quanto para o KVM. A partir disso, a energia média consumida por máquina virtual aumenta à medida que novas instâncias são adicionadas. Nesse contexto, o KVM apresentou um desempenho superior em relação ao Xen para ambientes que contém de 2 a 6 máquinas virtuais. Acima ou abaixo

desses valores, o DomU se mostra mais eficiente. Deve-se notar que há um grande ganho energético advindo da consolidação de servidores que compensa esse aumento na energia média consumida. Dentro desse contexto, na execução do *benchmark* Iozone em uma única máquina virtual, foi verificada a estratégia Race *to Idle*, a qual diz que é mais vantajoso finalizar determinada tarefa mais rapidamente – onerando na potência dissipada – do que permanecer executando-a em um nível menor de potência, já que, nesse caso, o consumo energético final pode ser maior.

É importante ressaltar que, quando submetidas a tarefas que requerem uso intensivo do processador, a utilização de *n* máquinas virtuais em um único sistema físico torna o ambiente mais energeticamente eficiente do que a utilização de *n* sistemas físicos independentes. No cenário de teste com operações de E/S, embora a energia média dissipada por máquina virtual tenha aumentado na medida em que se adicionavam mais instâncias virtuais, o ambiente final também pode ser considerado energeticamente eficiente, em virtude, principalmente, da questão de consolidação de servidores.

Este trabalho identificou uma série de aspectos relacionados ao consumo de energia e à virtualização que precisam ser melhor estudados como, por exemplo, o comportamento anômalo do KVM no *benchmark* Iozone. As operações de E/S parecem ter fundamental importância no consumo de energia, por isso deve-se entender os mecanismos empregados entre MV, hipervisores e sistemas nativos para compreender melhor o comportamento do sistema quando submetido a esse tipo de tarefa. Em especial, é importante analisar os aspectos de armazenamento, ou seja, sistemas de arquivos e de comunicação em rede. O estudo de consumo de redes virtuais estabelecidas entre *n* máquinas virtuais em uma mesma máquina física é particularmente interessante na área de *clusters* virtuais que executam aplicações MPI. Esses aspectos são possibilidades para trabalhos futuros.

Por fim, as técnicas atuais para reduzir o consumo de energia estão evoluindo e novas técnicas estão sendo estudadas de forma que haja uma melhora na eficiência energética geral dos sistemas. Contudo, a cultura dos usuários também deve evoluir. Hoje, dificilmente se escolhe um produto em razão de sua eficiência energética. Além disso, geralmente, as funções de redução de consumo são desabilitadas ou não utilizadas. Funcionários desperdiçam energia mantendo o computador da empresa ligado durante a noite. Até mesmo os *spams* são uma forma de desperdício. A McAfee estimou que, em 2008, cerca de 62 trilhões de spams foram enviados via e-mail. A energia consumida pelo envio dessas mensagens poderia abastecer cerca de 2.4 milhões de residências durante um ano inteiro [RUTH, 2009]. Deve-se, portanto, incentivar o desenvolvimento de medidas de redução de consumo energético em componentes computacionais para que possamos vivenciar um crescimento baseado no pilar da sustentabilidade.

REFERÊNCIAS BIBLIOGRÁFICAS

[BARROSO et al., 2007] L. A. Barroso; U. Holzle, The case for energy-proportional computing. IEEE Computer, p. 1-5, dez 2007.

[BRAGA, 2007] N. C. Braga. Entenda o fator de potência. Mecatrônica Fácil; Ano: 6; N° 34; Mai / Jun – 2007.

[BRAGA, 2010] N. C. Braga. Transformadores e fator de potência (EL104). Disponível em: <http://www.newtoncbraga.com.br/index.php/eletrotecnica/2193-el134.html>, acesso em: outubro 2011.

[BROWN et al., 2010] D. J. Brown; C. Ream, Toward Energy-Efficient Computing, ACM Queue, v.8, n.2, p. 1-13, fev. 2010.

[BUENO, 2009] H. Bueno. Virtualização, um Pouco de História, 2009. Disponível em <http://hbueno.wordpress.com/2009/04/29/virtualizacao-um-pouco-de-historia/>, acesso em outrubro de 2011.

[CARISSIMI, 2009] CARISSIMI, A. S. Virtualização: conceitos e aplicações em processamento paralelo, 2009.

[CARISSIMI et al., 2010] A. S. Carissimi, C. F. R. Geyer, N. Maillard, P. O. A. Navaux, G. G. H. Cavalheiro, M. L. Pilla, A. C. Yamin, A. S. Charão, B. Stein, C. A. F. De Rose, L. G. L. Fernandes, T. C. Ferreto, A. Zorzo. Energy-Aware Scheduling of Parallel Programs. Conferencia Latino Americana de Computación de Alto Rendimiento (CLCAR'2010). 2010, Gramado, Brasil.

[CPU WORLD, 2012]. Intel Core i3-540.Disponível em <http://www.cpu-world.com/CPUs/Core_i3/Intel-Core%20i3-540%20CM80616003060AE%20(BX80616I3540%20-%20BXC80616I3540).html>, acesso em junho de 2012.

[FAN et al., 2002] X. Fan, C. S. Ellis, A. R. Lebeck, Memory Controller Policies for DRAM Power Management, 2002.

[GARRET, 2008] M. Garret. Powering Down, ACM Queue, v.5, n.7, p. 17-21, jan. 2008.

[GUNARATNE et al., 2005] C. Gunaratne, K. Christensen, B. Nordman. Managing Energy Consumption Costs in Desktop Pcs and Lan Switches With Proxying, Split TCP Connections, and Scaling of Link Speed. Int.J. Network. Manag., v. 15, n. 5, p. 297-310, 2005

[HOPPER, 2009] J. Hopper. Reduce Linux power consumption, Part 1: The CPUfreq subsystem. Disponível em <http://www.ibm.com/developerworks/linux/library/l-cpufreq-1/index.html>, acesso em: outubro 2011.

[IOZONE, 2006] IOzone Filesystem Benchmark. Disponível em <http://www.iozone.org/>, acesso em: março 2012.

[IPERF, 2010] IPERF – The Easy Tutorial. Disponível em <http://openmaniak.com/iperf.php>, acesso em: março 2012.

[JONES, 2007] T. M. Jones. Discover the Linux Kernel Virtual Machine. Disponível em <http://www.ibm.com/developerworks/linux/library/l-linux-kvm/>, acesso em: agosto 2011.

[KVM, 2011] Kernel Based Virtual Machine. Disponível em: <http://www.linux-kvm.org/page/Main_Page> , acesso em : agosto 2011.

[LESS WATT, 2011] Optimizing for Power Performance. Disponível em <http://www.lesswatts.org/documentation/sw-silicon-features/>, acesso em: outubro 2011.

[NAKAJIMA, 2007] J. Nakajima. Hybrid Virtualization – The Next Generation of XenLinux. Disponível em < http://www.valinux.co.jp/documents/tech/presentlib/2007/2007xenconf/Intel.pdf>, acesso em: março 2012.

[NEWING, 2010] R. Newing. Powerful Argument for cutting IT energy consumption. Financial Times – Special Report: Green Innovation and Design, 16-09-2010, p. 2.

[NI, 2009]. Virtualization Basics. Disponível em <http://zone.ni.com/devzone/cda/tut/p/id/8708>, acesso em: março 2012.

[QEMU, 2012] QEMU. Disponível em <http://wiki.qemu.org/Main_Page>, acesso: abr. 2012.

[REN21, 2011] Renewables 2011: Global Status Report. Disponível em <http://www.ren21.net/Portals/97/documents/GSR/GSR2011_Master18.pdf>, acesso em: outubro de 2011.

[ROBIN et al., 2000] Robin, J.S.; Irvine, C.E. Analysis of the Intel Pentium.s Ability to Support a Secure Virtual Machine Monitor. Proc. 9a USENIX Security Simposium, [S.l.:s.n] 2000.

[RUTH, 2009] S. Ruth. Green IT - More Than a Three Percent Solution? IEEE Internet Computing, v.13 n.4 p. 74-78, jul. 2009.

[SAXE, 2010] E. Saxe, Power-Efficient Software, Queue, V.8, N.1, p.44-48, jan. 2010

[SPEC, 2011] SPEC CPU2006. Disponível em <http://www.spec.org/cpu2006/>, acesso em: março 2012.

[SMITH et al. 2005] J.E. Smith, R. Nair .The architecture of virtual machines. IEEE Computer, v.38, n.5, p. 32-38, 2005.

[SALGADO, 2011] G. T. Salgado. Estudo sobre o Impacto Energético de Máquinas Virtuais em um Sistema Computacional Físico, 2011.

[SULEMAN, 2011] A. Suleman. Do multicores save energy? Not really. Disponível em < http://www.futurechips.org/chip-design-for-all/a-multicore-save-energy.html>, acesso em: março 2012.

[VMWARE, 2011] VMWARE White Paper, Understanding Full Virtualization, Paravirtualization and Hardware Assist. Disponível em < http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf >, acesso em: ago. 2011.

[XEN, 2011] What is Xen? Disponível em <http://www.xen.org/files/Marketing/WhatisXen.pdf>, acesso em: setembro 2011.

ANEXO <ARTIGO TG1>

Virtualização e Consumo de Energia

João Paullo Vieira de Almeida[1], Alexandre da Silva Carissimi[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{jpvalmeida,asc}@inf.ufrgs.br

Resumo. *Nos últimos anos, o conceito de Computação Verde vem ganhando espaço, dando luz a diversas ferramentas e técnicas com o objetivo de aumentar a eficiência energética dos componentes de um computador. A virtualização pode aumentar a eficiência energética de sistemas computacionais, ao garantir uma utilização com menos desperdícios de hardware. Este trabalho propõe estudar o impacto energético resultante da execução de máquinas virtuais em uma máquina física, utilizando ferramentas bastante difundidas no mercado.*

1. Introdução

Com o incessante aumento da população mundial, a demanda por energia elétrica torna-se cada vez maior. Em 2009, estima-se que 84% da energia elétrica produzida mundialmente foi obtida por meio de fontes não renováveis [REN21, 2011], que acarretam prejuízos ao meio ambiente. Atualmente, em virtude dos alertas cada vez mais urgentes sobre as consequências que esse aumento na demanda pode ocasionar e dos dados relativos à emissão de gases na atmosfera, governos e empresas estão, como nunca estiveram, dando maior atenção à necessidade de aumentar sua eficiência energética.

Na área da computação, o consumo de energia sempre foi uma questão importante para fabricantes de sistemas embarcados ou de computadores portáteis. A duração das baterias e a dissipação de calor de tais sistemas são aspectos imprescindíveis para o sucesso desses produtos; por esse motivo, diversos estudiosos e desenvolvedores propuseram técnicas para reduzir o consumo energético em plataformas portáteis. Todavia, a constante demanda por melhor desempenho, aliada ao aumento do custo energético, estendeu o problema do consumo de energia aos *desktops* e aos servidores.

O conceito de *Green Computing* (Computação Verde) foi introduzido em 1992, quando a *US Environmental Protection Agency* (EPA) lançou o padrão *Energy Star* [RUTH, 2009]. Trata-se de um padrão fornecido a equipamentos eletrônicos energeticamente eficientes. Ao longo do tempo, o *Energy Star* tornou-se uma certificação respeitada e reconhecida mundialmente. Atualmente, empresas buscam atingir esse padrão para seus produtos.

De acordo com [GUNARATNE et al., 2005], um dispositivo energeticamente eficiente consome energia proporcionalmente à quantidade de trabalho que ele produz. Nesse contexto, quando equipamentos com essa característica apresentam um aumento na quantidade de potência dissipada, é observado, também, um aumento proporcional na quantidade de trabalho produzida por esse aparelho. Em sistemas computacionais, a quantidade de trabalho executada pode, superficialmente, ser representada pelo número de instruções executadas pelo processador. Dessa maneira, fabricantes buscam diminuir a relação energia por instrução executada com o objetivo de melhorar a eficiência energética do sistema.

Os sistemas computacionais apresentam baixas taxas de eficiência quando estão submetidos a uma carga pequena de tarefas. Servidores, por exemplo, raramente estão em estado de completa ociosidade (*idle)*, e dificilmente operam perto de sua capacidade máxima de operação. Ao invés disso, eles trabalham na maior parte do tempo entre 10% e 50% de seu nível de utilização máximo [BARROSO et. al., 2007]. Considerando que a relação energia consumida por trabalho produzido é bastante inferior em momentos de ociosidade do processador, a diminuição da quantidade de potência dissipada não se ajusta corretamente com a diminuição da quantidade de trabalho que o computador produz [BROWN et. al., 2010].

Nesse contexto, a virtualização surge como uma alternativa para aumentar a eficiência energética de computadores. Máquinas virtuais (MV) fornecem uma plataforma abstrata de execução, independente da máquina física. Desta forma, diversas máquinas virtuais podem executar em uma mesma máquina física, garantindo uma maior utilização dos recursos dessa máquina e liberando outras máquinas físicas (possibilitando assim a sua desativação). Essa configuração é especialmente interessante em ambientes de *data centers*, onde diversas máquinas físicas provêm diferentes serviços. Ao se executar os serviços em máquinas virtuais, é possível desacoplá-lo da máquina física, permitindo-se que diversos serviços sejam executados em um mesmo servidor (quando a demanda no *datacenter* for pequena) e garantindo-se ainda um sistema robusto, capaz de suportar picos de ocupação dos servidores.

O objetivo deste trabalho é estudar, o impacto energético resultante da execução de máquinas virtuais em uma máquina física. A intenção é validar a ideia mencionado no início desta seção que diz que a adição de novas máquinas virtuais aumenta a eficiência energética da máquina física. Para tal, serão utilizadas duas das ferramentas mais disseminadas atualmente: Xen [Xen, 2011] e KVM [KVM, 2011]. Como carga de processamento pretende-se utilizar *benchmarks* padrões como, por exemplo, Spec CPU2006, NAS e Iperf.

O artigo é organizado da seguinte forma: a seção 2 descreve trabalhos relacionados sobre consumo de energia. A seção 3 revisa os conceitos físicos básicos como consumo, energia e potência. Logo após, a seção 4 apresenta estratégias para reduzir o consumo de energia. A seção 5 consiste na apresentação dos principais elementos da virtualização computacional. Por fim, é apresentada a proposta do trabalho de conclusão e seu cronograma.

2. Trabalhos Relacionados

Diversos trabalhos abordam o tema do consumo de energia em plataformas computacionais. Em [CARISSIMI et al, 2010], há um estudo sobre o impacto no consumo de energia em diferentes tipos de escalonamento de aplicações paralelas em sistemas distribuídos (cluster com *p* CPUs). A grande questão é balancear os ganhos em tempo de execução devido ao alto grau de paralelismo com as perdas energéticas ocorridas em virtude do mesmo paralelismo. Dependendo do tipo de tarefa a ser executada, a diminuição da frequência de processamento também se mostra como uma forma eficiente de redução de consumo de energia. Foi concluído que o custo energético para executar uma aplicação paralela em hardware apropriado pode se manter razoavelmente constante se o *speedup* (valor que diz quanto um algoritmo paralelo é mais rápido que seu correspondente algoritmo sequencial) é linear ou próximo disso, a fim de compensar o aumento linear de CPUs (ou núcleos) que são utilizados.

Em [LEFÈVRE et al, 2010] e [ORGERIE et al., 2010], são realizados estudos sobre o consumo energético de arquiteturas Cloud, assunto altamente em voga atualmente. De acordo com [GARRETT, 2008], processadores atuais, com centenas de milhões de

transistores, podem gerar mais de 100 Watts de calor, sendo os maiores dissipadores de potência em uma plataforma computacional. Em [GROVER, 2003], é dito que, ao diminuirmos a voltagem de alimentação do sistema, a frequência de processamento deve ser diminuída também, ocasionado um aumento no tempo de execução das aplicações. Com isso, a redução da tensão de 1.3V para 1V, por exemplo, acarreta uma economia de energia de aproximadamente 40%.

De acordo com [BROWN et al, 2010], há duas técnicas básicas para gerenciamento de energia: a primeira leva em consideração que um dispositivo de hardware pode desativar alguns de seus componentes que não estão sendo utilizados. Quando esse componente precisa ser utilizado novamente, ele é reativado. A segunda abordagem busca modificar os parâmetros de desempenho dos dispositivos durante sua execução (*on the fly*), baseando-se na carga de trabalho que está sendo processada. Essas abordagens serão aprofundadas na seção 4.

Em suma, os assuntos relativos ao consumo de energia e à eficiência energética de dispositivos são tópicos vastamente debatidos atualmente.

3. Consumo de Energia

Energia é uma grandeza física que representa a capacidade de se realizar trabalho. Ela é medida em Joules (J) ou em quilowatt-hora (1 kWh = 3600 kJ). Potência é a taxa com que a energia é transferida, medida em Watt (W = Joules por segundo). A relação entre energia e potência é:

$$P = \frac{dE}{dt} \qquad (equação\ 3.1)$$

Sendo *P* a potência, *E* a energia e *t* o tempo.

Em sistemas elétricos, a potência pode ser definida como o trabalho realizado pela corrente elétrica em um determinado intervalo de tempo. Em um sistema de corrente contínua (*Direct Current* – DC) no qual a tensão (V), medida em Volts, e a corrente (I), medida em Ampères, se mantém constantes durante um dado período, a potência transmitida é dada por:

$$P = V.I \qquad (equação\ 3.2)$$

Sabendo que $V = RI$, podemos chegar à expressão:

$$P = I^2R \qquad (equação\ 3.3)$$

Sendo R a resistência do circuito, medida em Ohm (Ω).

Para o caso de sistemas de corrente alternada (*Alternating Current* – AC), a tensão é comumente expressa em V_{RMS} (*Root Mean Square*, em inglês). A corrente, por sua vez, é I_{RMS}. Em sistemas desse tipo, a corrente *I(t)* – e, portanto, a potência instantânea – são funções dependentes do tempo, logo, a equação 3.3 precisa ser estendida para refletir esse fato. Se a função é periódica como ocorre, por exemplo, na rede elétrica pública de transmissão, a análise deve ser feita em relação à potência média dissipada ao longo do tempo. Logo:

$$P_{média} = < I(t)^2R > = R. < I(t)^2 > = \ R.{I_{RMS}}^2 \qquad (equação\ 3.4)$$

Ou seja, o valor RMS, I_{RMS}, da função *I(t)* é o valor constante que leva à mesma dissipação de potência que a dissipação de potência média ao longo do tempo da corrente *I(t)*.

Podemos, também, estender a relação para um sistema com tensão não constante, *V(t)*, com valor RMS expresso por V_{RMS}. Temos:

$$P_{média} = \frac{{V_{RMS}}^2}{R} \qquad (equação\ 3.5)$$

Ao igualarmos as equações 3.4 e 3.5, obtemos:

$$P_{média} = V_{RMS} I_{RMS} \qquad (equação\ 3.6)$$

É importante ressaltar que as equações aqui obtidas são dependentes dos sistemas serem puramente resistivos (por exemplo: chuveiros, lâmpadas incandescentes, ferros de passar roupa), o que significa que não há cargas reativas, tais como capacitores e indutores, que são capazes de não só dissipar energia, mas também de armazená-la. Como exemplo de cargas indutivas, há os motores e os transformadores. Por sua vez, cargas capacitivas podem ser representadas por lâmpadas fluorescentes e computadores.

Quando uma tensão senoidal é aplicada numa carga resistiva, a corrente circulante pela carga acompanha instantaneamente as variações de tensão, como mostra a figura 3.1.

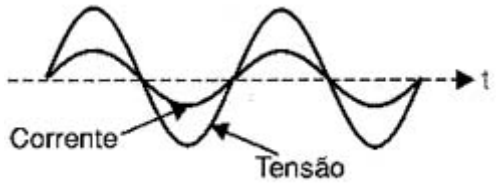


Figura 3.1: Corrente e tensão estão, normalmente, em fase em um dispositivo puramente resistivo alimentado por corrente alternada. Adaptado de [BRAGA, 2010].

Contudo, na maioria dos casos reais, o sistema não se comporta como uma resistência pura. Adicionando-se componentes reativos ao sistema, o fluxo de potência flutua, ou seja, a corrente pode fluir tanto em um sentido quanto em outro ao longo do tempo. Nesse caso, a corrente não acompanha as variações de tensão instantaneamente, havendo um retardo ou um adiantamento, conforme ilustra a figura 3.2.

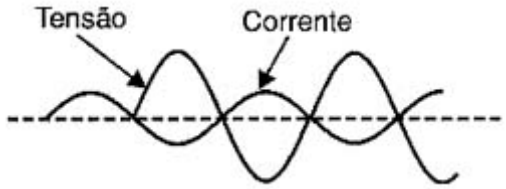


Figura 3.2: Corrente e tensão não estão em fase quando há presença de componentes reativos. Adaptado de [BRAGA, 2010].

Nesse caso, é necessário introduzir alguns conceitos como potência real ou potência ativa (P), potência reativa (Q), potência complexa (S) e potência aparente (|S|). A potência real (ativa), medida em Watt, representa a energia gasta em determinado espaço de tempo. A potência reativa, medida em volt-ampère reativo (var), representa a porção do fluxo de potência devido à energia armazenada que retorna à fonte a cada ciclo. Por fim, temos a potência complexa, medida em volt-ampère (VA), e a potência aparente, também medida em volt-ampère (VA), que é o valor absoluto da potência complexa e representa o produto da corrente pela tensão do circuito. O esquema descrito está ilustrado na Figura 3.3.

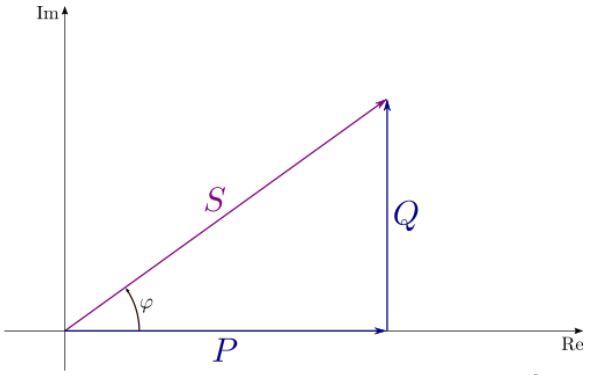


Figura 3.3: A potência complexa (S) é o vetor soma das potências real (P) e reativa (Q). A potência aparente (|S|) é a magnitude da potência complexa.

O ângulo φ formado pela potência complexa e pela potência real é usado para indicar o fator de potência (FP), que varia entre 0 e 1, e é dado por:

$$FP = cos\varphi \qquad (equação\ 3.7)$$

O melhor aproveitamento num sistema de corrente alternada ocorre quando o ângulo de defasagem é zero – ou seja, cosseno igual a 1. Nesse caso, 100% da energia aplicada é convertida em trabalho. Na prática, no entanto, equipamentos raramente atingem esse padrão. Um fator de potência baixo significa que energia reativa está sendo gerada e não é aproveitada [BRAGA, 2007]. Se esse desperdício ocorre em larga escala, as concessionárias de energia têm grandes prejuízos. Em virtude disso, em 1993, o extinto DNAEE (Departamento Nacional de Águas e Energia Elétrica), por meio da portaria número 1569, determinou que as instalações elétricas das unidades consumidoras devem ter um fator de potência mínimo de 0,92. Se o fator de potência estiver abaixo desse valor, a fatura mensal de energia elétrica sofrerá um reajuste.

Com efeito, a medida de potência em Watt de um equipamento eletrônico determina a verdadeira potência que a companhia de energia elétrica deve fornecer para o dispositivo funcionar. Por outro lado, a grandeza volt-ampère (VA) é usada para dimensionar os fios e sistemas de proteção das instalações elétricas. A classificação Watt e VA é igual para alguns tipos de aparelhos, tal como as lâmpadas incandescentes. No entanto, para equipamentos eletrônicos, essas grandezas podem se diferenciar significativamente, com a medida em VA sendo sempre igual ou maior que a medida em Watt.

Em sistemas computacionais, como visto em [CARISSIMI et al, 2010], a energia utilizada para realização de um programa executado em um processador baseados em *switches* CMOS pode ser expressa como:

$$E_{op} = C * {V_{dd}}^2 * f * \left(\frac{N}{f}\right) + I_{fuga} * V_{dd} * \left(\frac{N}{f}\right) \qquad (equação\ 3.8)$$

Onde E_{op} é a energia, C a capacitância de chaveamento, V_{dd} é a voltagem na qual o circuito está operando, f é a frequência de operação, I_{fuga} a corrente de fuga e N o número de ciclos requeridos para executar o programa.

Observando a expressão 3.8, notamos que, para diminuir o consumo, podemos reduzir a voltagem de alimentação (V_{dd}). Contudo, por razões físicas, a frequência de processamento deve ser diminuída para que tal mudança seja possível. A redução da frequência faz com que o processador execute instruções em um ritmo menor, prejudicando o seu desempenho. Surge, então, um compromisso consumo energético versus desempenho.

4. Estratégias Básicas para Redução do Consumo de Energia

De uma forma simplificada, existem quatro maneiras de se trabalhar para reduzir o consumo de energia. Primeiramente, por meio de funcionalidades fornecidas pelos componentes de hardware, as quais podem ser divididas em dois grupos: as que exploram períodos de ociosidade e as que se aproveitam da possibilidade de se utilizar os recursos computacionais abaixo do seu limite. Os *idle states* e *frequency scaling*, respectivamente, ilustram essas duas abordagens. A primeira técnica defende que o dispositivo de hardware deve ter a capacidade de desativar seus componentes que não estão em uso. Em longos períodos de ociosidade, diferentes períodos de inatividade – *idle states* – podem ser definidos. Dessa forma, o dispositivo pode desativar gradualmente as suas funcionalidades. Essa estratégia considera que, se o dispositivo está a um longo tempo sem uso, ele provavelmente ficará ainda muito mais tempo sem ser utilizado. No entanto, deve-se mensurar o custo – em termos energéticos e temporais – da reativação do dispositivo. Quanto mais profundo o *idle state*, menos energia ele consome, todavia, o custo da reativação do componente é maior. Logo, o dispositivo deve permanecer sem uso tempo suficiente para que tais custos sejam compensados. Processadores, memórias e discos utilizam *idle state* para reduzir consumo energético.

A segunda técnica – *frequency scaling* – busca a redução do consumo à custa do desempenho. Como visto na equação 3.8, a diminuição da frequência de processamento reduz linearmente seu consumo de energia e permite, também, a redução da tensão de alimentação do processador. Dessa maneira, reduzir a frequência de processamento provoca um grande impacto da potência dissipada. Processadores modernos apresentam ferramentas de hardware que permitem variar a frequência de processamento. Essa variação pode ser feita sob demanda ou conforme alguma política definida pelo sistema operacional ou, ainda, pelo usuário.

A segunda maneira para reduzir o consumo de energia é em nível do sistema operacional, que faz uso das funcionalidades descritas anteriormente. O sistema operacional, por meio das instruções e das chamadas de sistema, se comunica com o processador e com os outros componentes físicos. Portanto, é ele que tem o controle sobre o hardware e detém todas as informações úteis para gerenciá-lo. Entre os métodos utilizados para reduzir o consumo energético do sistema operacional, destacam-se o *Dynamic Tick Model* e o *CPU Frequency Scaling*. A primeira técnica visa se beneficiar do fato de que o processador apresenta momentos de ociosidade. Do ponto de vista do processador, um estado de inatividade profundo – no qual mais partes do hardware são desativadas – otimizará o consumo energético apenas se a CPU ficar tempo suficiente nesse estado. Logo, um escalonador que gera interrupções contínuas a cada preempção (normalmente 100 vezes por segundo) pode ser bastante inconveniente quando a CPU está em estados de baixo consumo. Ao invés de utilizar um temporizador estático, o núcleo do sistema operacional, por meio do seu escalonador, pode observar os processos em execução e disparar interrupções quando considerar mais conveniente. Idealmente, um temporizador dinâmico permite ao processador ficar quase inteiramente desativado até que um usuário interaja com o sistema (interrupção de hardware).

A segunda técnica está presente a partir do kernel 2.6.0 do GNU/Linux. É a infraestrutura chamada *CPUfreq*. Quando processadores operam em velocidades menores de relógio, eles consomem proporcionalmente menos potência e dissipam menos calor. O gerenciamento dinâmico da velocidade do relógio por meio do *CPUfreq* fornece um controle para obrigar o sistema a consumir menos potência quando não operado na sua capacidade máxima [HOPPER, 2009]. Essa solução baseia-se no conceito de governadores. Um governador é um algoritmo que calcula a frequência que ele considera adequada para a CPU

em um determinado momento. Além dos governadores, é possível selecionar, manualmente, uma frequência específica de processamento. A tabela 3.1 mostra exemplos de governadores.

Tabela 3.1: Governadores para CPU *Frequency Scaling*. Fonte: HOPPER, 2009.

Governador	Descrição
Performance	Coloca o processador na sua frequência máxima de operação. Busca máximo desempenho.
Powersave	Coloca o processador na sua frequência mínima de operação.
Userspace	Permite ao usuário ou a um programa definir uma frequência manualmente.
Ondemand	Verifica regularmente a taxa de utilização do processador. Compara essa taxa com um valor pré-definido. Se a taxa é inferior a esse valor, configura o processador na mínima frequência de operação. Quando a taxa é superior a esse valor, coloca o processador na sua frequência máxima de operação.
Conservative	Define dois valores: *up_threshold* e *down_threshold*. Verifica regularmente a taxa de utilização. Se a taxa é inferior a *down_threshold*, coloca a CPU para executar na primeira frequência mais baixa que a atual. Analogamente, se a taxa é superior a *up_threshold*, coloca a CPU na primeira frequência mais alta que a atual. É mais custoso computacionalmente que o Ondemand.

O terceiro fator importante para redução do consumo energético de um sistema computacional é o conjunto de aplicações que o usuário executa. Embora não tenham acesso direto ao hardware – sendo necessário o sistema operacional para realizar essa interação –, existem diversas técnicas e recomendações que um desenvolvedor pode seguir para tornar sua aplicação mais energicamente eficiente. A seguir estão destacadas três delas.

***Race to Idle*:** Baseia-se no fato de finalizar as tarefas no menor tempo possível, para retornar ao estado de ociosidade (*idle*) o mais rápido possível. Um exemplo simplificado [LESS WATT, 2011] diz que um processador típico consome 34 Watts quando executado na frequência máxima ($f_{máx}$), e 24 Watts quando executado em $f_{máx}/2$ e 1 Watts quando encontra-se no estado *idle*. Supondo que executamos uma aplicação que dura 0,25 segundos na velocidade máxima. Temos, portanto, que o consumo de energia durante 1 segundo é:

- Em $f_{máx}/2$: $0{,}5s * 24W + 0{,}5s * 1W = 12{,}5\,Joules$

- Em $f_{máx}$: $0{,}25s * 34W + 0{,}75s * 1W = 9{,}25\,Joules$

Embora o exemplo anterior seja simplificado, a ideia central é o que ocorre em geral em sistemas reais: é melhor executar o mais rápido possível para que se possa ficar mais tempo em estado *idle*.

Utilizar *buffers* maiores: Dados que são transferidos entre dispositivos de entrada e saída e a memória são armazenados em regiões de memória temporária - *buffers*. Ao definir o espaço de memória ocupado pela aplicação, os desenvolvedores devem criar *buffers* suficientemente grandes para minimizar o número de acessos ao dispositivo de entrada e saída. Uma aplicação que lê e executa faixas de um DVD, por exemplo, deve armazenar em *buffer* pelo menos vinte minutos de vídeo para ter uma boa eficiência energética [LESS WATT, 2011].

Desativar dispositivos após o uso: Uma aplicação, ao terminar o uso de um dispositivo de hardware, deve informar ao sistema operacional que não necessita mais daquele dispositivo. Se isso não for feito, mesmo que a aplicação seja terminada, o sistema operacional vai manter aquele dispositivo em estado de ativação, desperdiçando energia.

Por fim, a quarta maneira de se reduzir o consumo de energia é por meio da virtualização, a qual desempenha um papel essencial, por exemplo, na consolidação de servidores em *data centers*, que são locais com grande índice de consumo de energia. Estima-

se que, nos Estados Unidos, estas infraestruturas são responsáveis por 1.5% do consumo energético do país [VMWARE, 2011]. Um dos fatores mais importantes para esse cenário é o mau aproveitamento dos recursos de hardware. Tipicamente, por motivos que variam desde a heterogeneidade de clientes à segurança, os *data centers* são projetados para executar um único serviço – servidor HTTP, de e-mail e de arquivos, por exemplo – em cada máquina física. O nível de carga de uma infraestrutura de TI pode depender do dia do mês ou do horário. Um *data center* deve ser projetado de forma a garantir um bom nível de serviço o tempo todo. Em uma arquitetura em que há um serviço por máquina física, os recursos de cada servidor devem ser mensurados de forma a suportar os picos de utilização. Desta forma, visto que os recursos serão plenamente utilizados apenas em momentos de pico, o hardware passará a maior parte do tempo subutilizado. Além disso, como foi comentado na introdução deste artigo, em estado de baixa carga, a CPU apresenta baixa eficiência energética. Em outras palavras, a incapacidade da infraestrutura de acomodar dinamicamente as flutuações no seu nível de utilização gera desperdício de hardware e energético.

Uma boa solução para este problema é a utilização de *data centers* baseados em máquinas virtuais. A ideia é fornecer um serviço por máquina virtual, ao invés de um serviço por máquina física. Um servidor pode executar diversas máquinas virtuais, portanto diversos serviços. Desta forma, essa nova arquitetura define *n* serviços para cada servidor. Com vários serviços executando em uma mesma máquina física, pode-se atingir um alto nível de utilização do seu hardware. Além disso, graças à flexibilidade de ambientes virtualizados, pode-se responder dinamicamente às variações de carga no sistema. Máquinas virtuais podem ser migradas de uma máquina física a outra sem grandes custos computacionais. Deste modo, o sistema pode ser gerenciado de forma a distribuir eficientemente os serviços entre os servidores. Pode-se concentrar todos os serviços em alguns servidores, permitindo que os outros sejam desligados.

A utilização de virtualização na consolidação de servidores garante uma utilização energética eficiente por dois fatores: i) menos hardware é necessário, pois ele é utilizado mais eficientemente e ii) conforme a demanda, servidores podem ser desligados, poupando a energia da alimentação desses servidores e dos seus sistemas de refrigeração. Cada servidor virtualizado pode economizar 7000 kWh de energia e deixar de emitir quatro toneladas de dióxido de carbono por ano [VMWARE, 2011].

5. Virtualização

O principal objetivo da virtualização é prover uma camada de software que ofereça um ambiente completo de execução – sistema operacional, bibliotecas e aplicações –, equivalente ao de uma máquina física. Com isso, as máquinas virtuais garantem isolamento entre os diversos usuários da máquina real. Atualmente, em virtude do aumento do poder computacional dos processadores e da popularização dos sistemas distribuídos, a virtualização surgiu como uma ferramenta para explorar mais eficientemente os recursos computacionais. Hoje, entre os grandes interesses dessa tecnologia estão: segurança, confiabilidade, custo, encapsulamento de um estado de computação, tolerância a falhas, criação de um ambiente seguro para testes, possibilidade de se executar múltiplos sistemas operacionais, consolidação de servidores, redução do custo de gerenciamento e de energia.

A virtualização é implementada por meio de uma camada de software localizada logo acima do hardware. Essa camada é chamada de monitor de máquina virtual – MMV – ou hipervisor (*hypervisor*). O processo ou sistema operacional que executa sobre uma máquina virtual é denominado hóspede. A plataforma onde a máquina virtual executa é denominada hospedeiro ou sistema nativo.

O MMV fornece uma interface baseada no hardware no qual está sendo executado. Na virtualização completa ou total, a ideia principal é fornecer ao sistema operacional hóspede uma réplica do hardware subjacente. Para tal, o MMV deve implementar todos os componentes e os elementos relacionados ao hardware presentes na máquina física e que podem ser usados pela máquina virtual. O grande aspecto positivo dessa técnica é a transparência em relação às aplicações. Ou seja, um sistema operacional pode executar em uma máquina virtual desse tipo sem ser alterado. Contudo, experiências mostram que essa técnica apresenta uma sobrecarga grande ao sistema. Com efeito, o desempenho pode ser melhorado permitindo que uma interface modificada seja colocada logo acima do hardware. Essa técnica é chamada de paravirtualização. A grande desvantagem dela é a necessidade de alterar o sistema hóspede. Essa modificação pode ser indesejada ou, até mesmo, impossível, como no caso de sistemas de código proprietário como o Windows.

Na prática, há diversas ferramentas de virtualização disponíveis. Para este trabalho, foram escolhidas duas delas: Xen e KVM. O Xen é um monitor de máquina virtual desenvolvido em software livre, nos termos da GNU General Public License (GPL). O hipervisor Xen é compatível com várias arquiteturas, tais como, x86, x86_64, IA64, Itanium, Power PC e ARM. Ele suporta uma variedade de sistemas operacionais hóspedes como, por exemplo, GNU/Linux, FreeBSD, Solaris, Windows, etc.

Em relação à arquitetura Xen, ilustrada na figura 5.1, podemos destacar três componentes básicos: o hipervisor, o Dom0 – um domínio privilegiado – e os DomUs – múltiplos domínios não privilegiados. O hipervisor é a camada de abstração básica que se situa entre o hardware e o sistema operacional. Ele é o responsável pelo escalonamento de processos e pelo particionamento da memória entre as diversas máquinas virtuais executando no hardware nativo. O hipervisor abstrai não só o hardware para as máquinas virtuais, mas também controla a execução das máquinas virtuais como se elas compartilhassem um ambiente comum de processamento. O hipervisor não tem qualquer conhecimento sobre a rede, dispositivos de armazenamento externos, vídeo, ou qualquer outra função de E/S encontrada em um sistema computacional; ele é apenas uma interface para as requisições desses dispositivos.

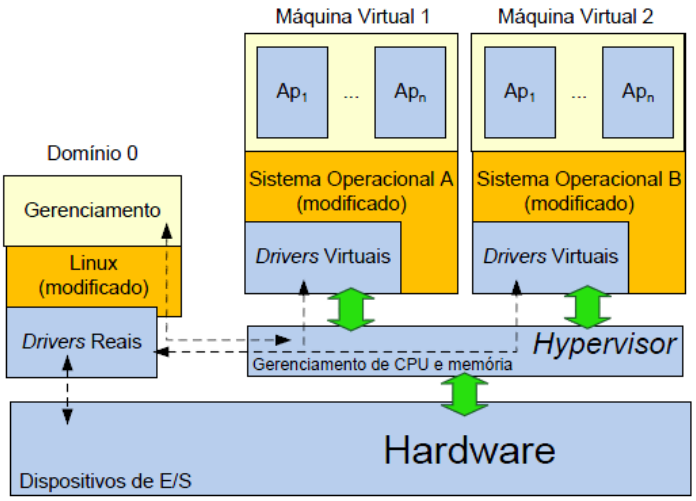


Figura 5.1: Arquitetura Xen. Adaptado de [CARISSIMI, 2009].

O domínio 0 (Dom0) é um sistema operacional modificado (tipicamente GNU/Linux) que atua como uma máquina virtual especial. Ele funciona como um gerenciador dos domínios U, efetuando tarefas como criar, iniciar, interromper, recomeçar e destruir. Somente

ele tem direitos para acessar os recursos físicos de E/S, tornando-o responsável por tratar as requisições de E/S (acesso à rede, ao disco, etc) efetuadas pelos outros domínios hóspedes, os DomUs. Os domínios Us, DomUs, são máquinas virtuais controladas pelo Dom0 e executam de forma independente umas das outras. Elas podem executar um sistema operacional modificado ou um sistema operacional não modificado com auxílio do hardware.

Por sua vez, a abordagem realizada pelo KVM (*Kernel Virtual Machine*) é transformar um núcleo Linux em um hipervisor simplesmente carregando um módulo no próprio núcleo. Esse módulo exporta um dispositivo chamado /dev/kvm, que ativa o modo convidado (*guest mode*, em adição aos dois modos de acesso padrão: núcleo e usuário). Com o KVM instalado, é possível iniciar sistemas operacionais hóspedes em espaço usuário. Cada sistema operacional hóspede é tratado como um processo comum do sistema operacional hospedeiro (ou hipervisor) e possui seu próprio espaço de endereçamento, que é mapeado quando o sistema operacional hóspede é instanciado (através do *kvm utility*).

Observando a figura 5.2, na base, encontra-se uma plataforma de hardware que tem a extensão de virtualização por hardware – atualmente, isso significa processadores Intel VT ou AMD-V. Logo acima da camada de hardware, está o hipervisor (o núcleo do Linux com o módulo KVM). Esse hipervisor se parece como um núcleo normal do Linux no qual podemos executar qualquer outra aplicação.

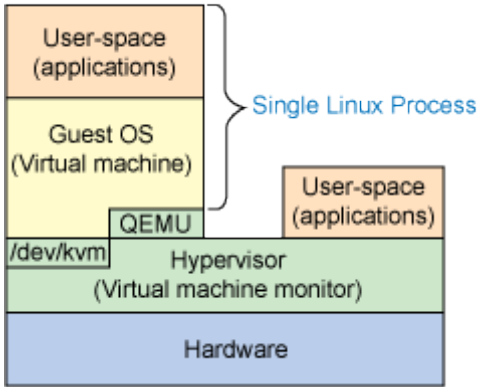


Figura 5.2: Arquitetura KVM [JONES, 2007].

É importante ressaltar que o KVM é uma parte de uma solução de virtualização. O processador fornece o suporte à virtualização diretamente. A memória é virtualizada por meio do *kvm utility*. Finalmente, E/S é virtualizada através de um processo QEMU ligeiramente modificado, que emula hardwares como dispositivos de rede, discos rígidos, placas de vídeo, etc.

O modo convidado, como o próprio nome sugere, é usado para execução do código do sistema operacional hóspede. Esse modo executa todos os códigos do sistema operacional hóspede, exceto código de E/S. Dentro dele, há os dois modos tradicionais, permitindo que uma aplicação executando como convidada possa utilizar, também, o modo núcleo, para execução dentro do núcleo, e o modo usuário, para o restante das aplicações, incluindo E/S, que é tratada independentemente. Qualquer pedido de E/S que um sistema operacional hóspede realiza são redirecionados para modo usuário para serem emulados pelo processo QEMU.

6. Proposta de Trabalho de Graduação

Mensurar o consumo energético das máquinas virtuais empregando dois hipervisores bastante comuns: Xen e KVM. Para atingir esse objetivo, a metodologia proposta consiste em empregar *benchmarks* padrões (Spec CPU2006, NAS, Iperf) para medir o consumo de energia nos seguintes padrões:

- Sistema operacional nativo;

- Executando sobre sistemas operacionais virtualizados utilizando as ferramentas Xen e KVM;

- Instanciando diversas máquinas virtuais sobre uma mesma máquina física e avaliando o consumo dessas *n* instâncias.

A obtenção das medidas se dará por meio de um multímetro digital acoplado ao cabo de alimentação da fonte da máquina física.

7. Cronograma

Para a realização do TG-2, as etapas previstas estão apresentadas na tabela 7.1.

Tabela 7.1: Cronograma TG-2.

	DEZ	JAN	FEV	MAR	ABR	MAI	JUN
Estudo e instalação do KVM e do Xen*****	X						
Execução de *benchmarks* sobre Xen		X	X				
Análise dos resultados - Xen			X				
Execução de *benchmarks* sobre KVM				X	X		
Análise dos resultados - KVM					X		
Consolidação final dos resultados					X		
Entrega e apresentação da monografia					X	X	X

***** Atividade já em curso

8. Referências Bibliográficas

[BARROSO et al., 2007] L. A. Barroso, U. Holzle. The case for energy-proportional computing. IEEE Computer, p. 1-5, dez 2007.

[BRAGA, 2007] N. C. Braga. Entenda o fator de potência. Mecatrônica Fácil; Ano: 6; N° 34; Mai / Jun – 2007.

[BRAGA, 2010] N. C. Braga. Transformadores e fator de potência (EL104). Disponível em: < http://www.newtoncbraga.com.br/index.php/eletrotecnica/2193-el134.html>, acesso em: outubro 2011.

[BROWN et al., 2010] D. J. Brown, C. Ream, Toward Energy-Efficient Computing, ACM Queue, v.8, n.2, p. 1-13, fev. 2010.

[CARISSIMI, 2009] A. S. Carissimi. Virtualização: conceitos e aplicações em processamento paralelo, 2009.

[CARISSIMI et al., 2010] A. S. Carissimi, C. F. R. Geyer, N. Maillard, P. O. A. Navaux, G. G. H. Cavalheiro, M. L. Pilla, A. C. Yamin, A. S. Charão, B. Stein, C. A. F. De Rose, L. G. L. Fernandes, T. C. Ferreto, A. Zorzo. Energy-Aware Scheduling of Parallel Programs.

Conferencia Latino Americana de Computación de Alto Rendimiento (CLCAR'2010). 2010, Gramado, Brasil.

[GROVER, 2003] A. Grover. Modern System Power Management. ACM Queue, v. 1, n. 7, p. 66-72, out. 2003.

[GARRET, 2008] M. Garret. Powering Down, ACM Queue, v.5, n.7, p. 17-21, jan. 2008.

[GUNARATNE et al., 2005] C. Gunaratne, K. Christensen, B. Nordman. Managing Energy Consumption Costs in Desktop Pcs and Lan Switches With Proxying, Split TCP Connections, and Scaling of Link Speed. Int.J. Network. Manag., v. 15, n. 5, p. 297-310, 2005.

[HOPPER, 2009] J. Hopper. Reduce Linux power consumption, Part 1: The CPUfreq subsystem. Disponível em <http://www.ibm.com/developerworks/linux/library/l-cpufreq-1/index.html>, acesso em: outubro 2011.

[JONES, 2007] T. M. Jones. Discover the Linux Kernel Virtual Machine. Disponível em <http://www.ibm.com/developerworks/linux/library/l-linux-kvm/>, acesso em: agosto 2011.

[KVM, 2011] Kernel Based Virtual Machine. Disponível em: <http://www.linux-kvm.org/page/Main_Page> , acesso em : agosto 2011.

[LESS WATT, 2011] Optimizing for Power Performance. Disponível em <http://www.lesswatts.org/documentation/sw-silicon-features/>, acesso em: outubro 2011.

[LEFÈVRE et al., 2010] L. Lefèvre, A. C. Orgerie. Designing and evaluating an energy efficient Cloud. Journal of SuperComputing, Special issue on Emerging Research in Parallel and Distributed Computing, 51:352-373, March 2010.

[ORGERIE, et al., 2010] A. C., Orgerie, L. Lefèvre, J. P. Gelas. Demystifying Energy Consumption in Grids and Clouds. Work in Progress in Green Computing (WIPGC) Workshop, in conjunction with the first International Green Computing Conference (IGCC), Chicago, USA, pages 335-342, August 2010.

[REN21, 2011] Renewables 2011: Global Status Report. Disponível em <http://www.ren21.net/Portals/97/documents/GSR/GSR2011_Master18.pdf>, acesso em: outubro de 2011.

[RUTH, 2009] S. Ruth. Green IT - More Than a Three Percent Solution? IEEE Internet Computing, v.13 n.4 p. 74-78, jul. 2009.

[VMWARE, 2011] VMWARE White Paper, How VMware Virtualization Right-sizes IT Infrastructure to Reduce Power Consumation. Disponível em < http://www.vmware.com/files/pdf/WhitePaper_ReducePowerConsumption.pdf>, acesso em: nov. 2011.

[XEN, 2011] What is Xen? Disponível em <http://www.xen.org/files/Marketing/WhatisXen.pdf>, acesso em: setembro 2011.