Medindo e Modelando o Desempenho de Aplicações em um Ambiente Virtual

Fabrício Benevenuto, Cesar Fernandes, Matheus Caldas, Virgílio Almeida, Jussara Almeida

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais
Belo Horizonte – MG – Brasil

{fabricio, cesar, mtcs, virgilio, jussara}@dcc.ufmg.br

Abstract. *Virtual machines (VMs) provide a suitable environment to consolidate multiple services into few physical machines, reducing costs of ownership and management. The main questions before the migration of an IT infrastructure to a virtual environment concern the performance of the applications and services on the new environment. This paper proposes a framework for predicting performance of applications on virtual systems. We propose performance models and validate the models experimentally through a case study of a Web server running on Linux, which is migrated to a Xen environment. Moreover, we provide a series of performance analysis of applications running on the Xen virtual environment. Our results highlight the causes of an extra overhead due to VMs sharing the same CPU.*

Resumo. *Máquinas Virtuais (VMs) oferecem um ambiente adequado para consolidar múltiplos serviços em poucas máquinas, reduzindo custos de aquisição e gerenciamento. As principais questões antes da migração de uma infraestrutura de TI para um ambiente virtual estão ligadas ao desempenho das aplicações e serviços no novo ambiente. Este artigo propõe um arcabouço para prever o desempenho de aplicações em ambientes virtuais. Nós propomos modelos de desempenho e validamos os modelos experimentalmente através de um estudo de caso de um servidor Web executando em Linux, que é migrado para o ambiente Xen. Além disso, provemos uma série de análises de desempenho de aplicações executando no ambiente virtual Xen. Nossos resultados destacam as causas de um* overhead *extra devido ao compartilhamento da CPU pelas VMs.*

1. Introdução

Virtualização é uma técnica de particionar recursos de uma máquina em múltiplos ambientes, criando um novo nível de indireção entre os recursos físicos e as aplicações. Recentemente, virtualização tem recebido um interesse notável, sendo utilizada como uma forma de consolidar múltiplas aplicações em um número pequeno de servidores. De fato, consolidar servidores pode ajudar organizações de TI a reduzir complexidade e custos de gerenciamento, aquisição e consumo de energia, provendo um ambiente flexível para a execução de serviços. Com tantas vantagens é comum vermos organizações de TI interessadas em migrar seus serviços para plataformas consolidadas que utilizam virtualização.

Entretanto, antes de migrar aplicações de máquinas físicas para ambientes virtualizados, é necessário estimar o desempenho que as aplicações terão no novo ambiente. Prever o número de servidores necessários em um ambiente virtual, capazes de suportar a execução dos serviços oferecidos e respeitar acordos de nível de serviço (SLA) são assuntos importantes que precisam ser abordados antes da migração. Além disso, definir qual a plataforma de hardware mais adequada para ambiente virtual e sua melhor configuração são importantes passos antes da aquisição da nova plataforma e da migração para o ambiente virtual. Nesse contexto, existe uma necessidade atual por ferramentas e tecnologia para prever desempenho, provendo informações necessárias para a alocação de recursos e determinação da configuração ótima do sistema.

Este trabalho dá um importante passo nesta direção. Nós propomos modelos analíticos para prever o desempenho que aplicações, atualmente em execução em ambientes reais, vão adquirir se forem migradas para ambientes virtuais. Nossos modelos consideram como parâmetros métricas obtidas de aplicações em execução em sistemas Linux e estimam o desempenho destas aplicações no ambiente virtual Xen [5]. Os modelos são validados através de um estudo de caso sobre um servidor Web que é migrado de um ambiente real para um virtual. Particularmente, observamos a existência de interferência entre máquinas virtuais (VMs) quando elas compartilham a mesma plataforma. Nossos resultados quantificam as principais causas desse *overhead* extra e mostra que ele pode ser significativo.

O restante do artigo está organizado da seguinte forma. A próxima seção apresenta trabalhos relacionados, e a seção 3 discute a arquitetura do Xen. A seção 4 apresenta nosso arcabouço para previsão de desempenho e resultados experimentais para cada abordagem adotada no modelo são apresentados na seção 5. Um estudo de caso é apresentado na seção 6 e, finalmente, a seção 7 conclui e apresenta direções para trabalhos futuros.

2. Trabalhos Relacionados

Recentemente, virtualização se popularizou novamente causando o surgimento de novos ambientes [21] [5] [22] e o desenvolvimento de hardware especial para esses ambientes [20] [19]. Entretanto, consolidar servidores em ambientes virtuais não depende somente de desenvolver e aprimorar a capacidade da plataforma de virtualização, mas também de ferramentas com ênfase em planejamento de capacidade.

O problema de prever desempenho em ambientes virtuais não é totalmente novo. Virtualização teve suas raízes nos anos 70, quando um dos sistemas mais populares era o IBM VM/370. Como uma conseqüência de sua popularidade, o desempenho do VM/370 foi estudado [2] e modelos analíticos foram desenvolvidos para estimar métricas de desempenho como tempo de resposta e utilização de CPU nesse ambiente [3]. Como uma evolução desses esforços, uma ferramenta de prever desempenho foi criada [4]. A principal motivação para o VM/370 era aumentar o nível de compartilhamento de mainframes. Com a evolução da computação, ambientes virtuais também evoluíram para lidar com novos desafios como segurança, consumo de energia, alto custo administrativo, etc. Nesse contexto, nosso trabalho tem como objetivo prever o desempenho de aplicações consolidadas nesse renovado e muito mais complexo cenário de virtualização.

Existem vários artigos que abordam alocação dinâmica de recursos em ambientes virtuais [10], [17]. Esses trabalhos, diferentemente do nosso, analisam aplicações já implantadas em um ambiente virtual. Consequentemente, os parâmetros de entrada para os modelos podem ser obtidas diretamente do ambiente virtual. Nosso modelo possui uma

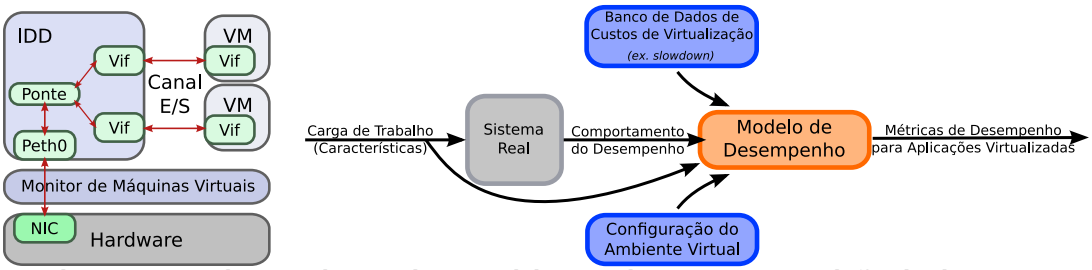


Figura 1. Arquitetura do Xen (esquerda) e arcabouço para previsão de desempenho (direita)

abordagem diferente pois, utiliza como entrada medições realizadas em ambiente real e prevê o desempenho das mesmas aplicações em um ambiente virtual.

Poucos trabalhos abordam alocação de recursos em ambientes virtuais tendo como foco a migração do ambiente não virtual. Recentemente, modelos baseados em teoria de filas foram propostos, sem validação, para prever o desempenho de aplicações executando em ambientes virtuais [12, 11]. Além disso, modelagem de um servidor Web simples foi proposta e validada com experimentação usando o Xen [6]. Esses trabalhos provêm apenas equações simplificadas, sem fornecer parâmetros importantes de configuração (ex. cap) ou capturar características reais da camada de virtualização, diferentemente do nosso trabalho.

3. O Ambiente Virtual Xen

Xen é um monitor de máquinas virtuais (VMM) que permite múltiplas instâncias de sistemas operacionais executarem concorrentemente em uma única máquina física [5]. O Xen utiliza paravirtualização, onde o VMM pode ser acessado através de uma máquina virtual levemente modificada em relação ao hardware. A figura 1 (esquerda) mostra a arquitetura do Xen. Cada aplicação executando em um SO acessa dispositivos de hardware através de uma VM especial e com acessos privilegiados ao hardware chamada IDD (*isolated driver domain*). As outras VMs executam dispositivos simplificados que comunicam com o IDD para acessar os verdadeiros dispositivos de hardware. Uma VM acessa o hardware indiretamente através de um dispositivo virtual (Vif) conectado ao IDD. Para evitar cópia de dados, referências às páginas são transferidas através deste dispositivo ao invés dos verdadeiros dados de entrada e saída [9].

Note que a CPU que executa o IDD pode ser um potencial ponto de contenção do sistema e, consequentemente, é capturada em nossos modelos. Desde que a Intel anunciou o desenvolvimento de uma nova tecnologia capaz de prover acesso direto ao hardware para VMs [8], acreditamos que o processamento do IDD possa ser bastante reduzido no futuro. Sendo assim, optamos por capturar o IDD nos nossos modelos como uma forma de definir um modelo geral.

Outro importante ponto a ser modelado é a quantidade de recursos alocados para cada VM. O escalonador de créditos do Xen [23] utiliza um mecanismo de créditos de tempo de CPU, provendo um balanceamento de carga automático entre CPUs virtuais e físicas de uma plataforma SMP. O parâmetro cap deste escalonador permite a definição da porção máxima de todas as CPUs disponíveis que cada VM pode utilizar. Por exemplo, em um cenário com 2 VMs e uma máquina de 4 CPUs, se ajustarmos o cap das duas VMs para 50 e 350 respectivamente, significa que a primeira VM pode atingir 50% da utilização de uma única CPU enquanto que a outra VM pode utilizar o restante dos recursos. O escalonador de créditos é o padrão do Xen desde a versão 3.0.4. Os escalonadores mais populares

antecessores do escalonador de crédito, ainda estão disponíveis, mas serão futuramente removidos do código do Xen [23].

4. Arcabouço para Previsão de Desempenho

Nosso arcabouço para previsão de desempenho é baseado em modelos de filas de valores médios e possui uma arquitetura conceitual ilustrada na figura 1 (direita). O modelo possui três entradas. Uma delas é relativa ao desempenho da aplicação de interesse em um ambiente não virtual (ex. número de requisições, demanda de recursos). Outra entrada é de um banco de dados que possui conhecimento armazenado do *overhead* da virtualização. Esse custo da virtualização é medido em um ambiente base para diferentes aplicações e sistemas e depois utilizado como entrada nos modelos. A última entrada é a configuração do ambiente virtual como a alocação de recursos para cada VM e o *speedup* da nova plataforma de hardware onde o ambiente virtual será executado, relativo à plataforma do sistema não virtual. Os resultados calculados pelo modelo são tempo de resposta, taxa de saída de requisições e utilização de recursos. Em seguida, apresentamos os modelos de desempenho utilizados nesse arcabouço. Na seção 6 apresentamos um estudo de caso onde explicitamos cada passo do arcabouço proposto.

4.1. Modelando o desempenho em VMs

Nossa estratégia para construir modelos de desempenho consiste em definir um fator para representar o *overhead* introduzido pela camada de virtualização na execução de uma aplicação em uma VM. Esse *overhead* é causado basicamente por operações que necessitam de ser emuladas pela VM como, por exemplo, a soma de verificação do TCP, normalmente feito pelo hardware. Além disso, existe o *overhead* de operações de entrada e saída. Nesse último caso, as VMs não possuem acesso direto ao hardware, consumindo tempo de CPU do IDD. A tabela 1 apresenta uma breve descrição dos símbolos utilizados nos modelos.

Como premissas do nosso modelo, assumimos que cada VM executa uma aplicação de classe diferente, onde uma classe de aplicações representa um tipo de aplicação para o qual o overhead da virtualização foi calculado. As cargas geradas no sistema por aplicações de uma mesma classe devem possuir características semelhantes (ex. distribuição do tamanho, tempo entre chegadas, etc). Esse overhead é calculado em uma plataforma de hardware base, para o qual se sabe o *speedup* em relação a outras máquinas. A definição de um sistema base é comum na comparação de diferentes hardwares e utilizada, por exemplo, pelo SPEC CPU [18]. A interferência, discutida na seção 5.3 não foi incluída em nossos modelos.

Sejam N VMs, $vm_1, vm_2 \cdots, vm_N$, que compartilham a mesma plataforma de hardware. Cada VM executa uma aplicação de classe diferente, onde uma classe representa um grupo de aplicações para o qual o overhead da virtualização é semelhante. A vm_0 representa o IDD e nossa abordagem para a modelagem do IDD é apresentada na seção 4.2. Para o *overhead* da virtualização relativo à parte da vm_i, definimos o *slowdown* de uma classe i de aplicações, S_k^i como:

$$S_k^i = \frac{B_k^{vm_i}}{B_k^i} \tag{1}$$

onde $B_k^{vm_i}$ é o tempo que o recurso k ficou ocupado para executar a aplicação na vm_i, e B_k^i é o tempo que o recurso k ficou ocupado para executar a mesma aplicação executando

SÍMBOLO	DESCRIÇÃO
N	# de VMs no ambiente virtual
K	# de recursos diferentes no ambiente virtual
N_{cpu}	# de CPUs disponíveis no ambiente virtual
B_k^i	Tempo ocupado do recurso k para a classe i no ambiente real
D_k^i	Demanda do recurso k para a classe i no ambiente real
λ^i	Taxa de chegada de requisições na vm_i
S_k^i	*Slowdown* da classe i no recurso k
P_k^i	*Speedup* da classe i no recurso k
cap_i	Porção do total de recursos que vm_i pode utilizar
$B_k^{vm_i}$	Tempo ocupado do recurso k para executar vm_i
$U_k^{vm_i}$	Utilização do recurso k pela vm_i do ponto de vista da vm_i
$D_k^{vm_i}$	Demanda do recurso k pela vm_i
$R_k^{vm_i}$	Tempo de residência médio do recurso k pela vm_i
cp^i	Tempo de CPU para o IDD processar um pacote da classe i
pr^i	Número de pacotes por requisição da classe i
$D_{CPU,i}^{vm_0}$	Demanda de CPU no IDD devido a atividade de E/S na vm_i
$U_{CPU}^{vm_0}$	Utilização de CPU no do IDD
$R_{CPU,i}^{vm_0}$	Tempo de Residência no IDD para a classe de aplicações i
$U_{CPU,i}^*$	Soma das utilizações de todas as VMs, exceto vm_i e o IDD
λ_{max}^i	Taxa máxima de chegada requisições

Tabela 1. Tabela de símbolos utilizados no modelo

em um ambiente real sobre o sistema base. Note que, S_k^i é um parâmetro pré-produzido, medido na plataforma de hardware base e pré-armazenado para ser utilizado no futuro. Baseado na equação (1) podemos calcular a demanda de serviço, $D_k^{vm_i}$, de cada recurso k para o ambiente virtual como:

$$D_k^{vm_i} = D_k^i \cdot \frac{S_k^i}{P_k^i} \tag{2}$$

onde D_k^i representa a demanda do recurso k para o sistema equivalente real executando a mesma aplicação que a vm_i, e P_k^i é o *speedup* da plataforma de hardware desejada para a execução da classe i de aplicações comparada com a plataforma não virtualizada onde D_k^i foi obtido. O *speedup* é uma variável essencial para o modelo porque permite aos usuários escolherem entre diferentes plataformas de hardware para implantar seus ambientes virtuais. Como um exemplo, de uma maneira menos precisa, porém mais prática, o *speedup* pode ser obtido através do resultado do benchmark de CPU da SPEC [18], calculado para um conjunto de plataformas dentre as quais o usuário pode escolher.

Seja $cap_0, cap_1, \cdots, cap_N$ a porção do total de recursos de CPU alocados para cada vm_i respectivamente, incluindo o IDD. O parâmetro cap_i pode assumir um valor que varia de 0 a N_{cpu}, onde N_{cpu} é o número de CPUs da plataforma de hardware e $cap_i > 0$. Como simplificação, restringimos $\sum_{i=0}^{N} cap_i = N_{cpu}$. Nesse contexto, a utilização da vm_i no recurso k pode ser obtida como:

$$U_k^{vm_i} = \frac{\lambda^i \cdot D_k^{vm_i}}{cap_i} \tag{3}$$

onde λ^i é a taxa de chegada requisições ao servidor. Baseado nas equações 2 e 3 podemos calcular o tempo de residência médio estimado, $R_k^{vm_i}$, de uma VM no recurso k. O tempo de residência médio representa todo o tempo gasto pela VM em um determinado recurso, incluindo tempo em fila e tempo de processamento. $R_k^{vm_i}$ é dado por:

$$R_k^{vm_i} = \frac{D_k^{vm_i}}{1 - U_k^{vm_i}} \tag{4}$$

Com base no tempo de residência de todos os recursos, podemos calcular o tempo de resposta de uma aplicação executando em uma VM. Estimar o tempo de resposta médio pode ser bastante útil para manter acordos do nível do serviço (SLA) estabelecidos ainda no ambiente real. O tempo de resposta da classe i de aplicações é a soma do tempo de residência médio na vm_i em todos os recursos k e do tempo de residência médio no IDD, discutido em seguida.

4.2. Modelando o Desempenho de Operações de E/S

O IDD é o componente do Xen responsável por executar operações de entrada e saída (E/S) para outras VMs. O tráfego de rede que chega e sai das VMs é percebido pelo IDD como pacotes que são trocados com as VMs através de pontes ou mecanismos de roteamento. Nossa abordagem para prever a utilização de CPU do IDD consiste em: 1) determinar o custo de processar um pacote, cp^i no ambiente virtual para a classe i de aplicações; 2) analisar a carga gerada pela aplicação alvo no ambiente real medindo o número médio de pacotes por requisição, pr^i para a classe i. A demanda de CPU do IDD, $D_{CPU,i}^{vm_0}$, para a classe i, é dada por:

$$D_{CPU,i}^{vm_0} = \frac{cp^i \cdot pr^i}{P_{CPU}^i} \tag{5}$$

onde P_{CPU}^i é *speedup* da classe i na CPU. O cp^i precisa ser previamente medido e armazenado. Na seção 5 são apresentadas algumas razões para definir cp^i como uma função da classe i de aplicações. Além disso, cp^i pode ser modificado de acordo com a troca de hardware, representado pelo *speedup* introduzido na equação 5.

A utilização de CPU do IDD é a soma das utilizações devido a cada uma das VMs executando no ambiente virtual. Ela pode ser expressa como:

$$U_{CPU}^{vm_0} = \frac{\sum_{i=1}^{N}(D_{CPU,i}^{vm_0} \cdot \lambda^i)}{cap_0} \tag{6}$$

Consequentemente, o tempo de residência médio no IDD para a classe de aplicações, $R_{CPU,i}^{vm_0}$, é uma função de $U_{CPU}^{vm_0}$ e sua equação é dada por:

$$R_{CPU,i}^{vm_0} = \frac{D_{CPU,i}^{vm_0}}{1 - U_{CPU}^{vm_0}} \tag{7}$$

4.3. Limites Assimtóticos

A análise de limites assintóticos pode ser bastante útil para determinar a carga que o sistema pode suportar enquanto provê tempos de respostas razoáveis (antes da saturação). Uma questão comum em análise de sistemas virtuais é *qual o valor máximo de taxa de chegada de requisições, λ_{max}^i, que um determinado serviço consegue suportar ao ser migrado para um ambiente virtual*. Esta questão tem uma resposta fácil que depende das

demandas de todos os recursos, do parâmetro do escalonador cap_i, e da taxa de chegada de requisições em cada VM, λ^i. Para uma dada aplicação, existem dois componentes candidatos a serem o gargalo no ambiente virtual, o IDD e a vm_i analisada. Para o caso no qual a vm_i é o gargalo, temos a demanda de serviço $D_k^{vm_i}$, a utilização $U_k^{vm_i}$, e a taxa de chegada de requisições, λ^i, relacionados por $\lambda^i = \frac{U_k^{vm_i} \cdot cap_i}{D_k^{vm_i}}$ para todos os recursos k. Por causa da utilização de qualquer recurso não exceder 100%, temos $\lambda^i \leq cap_i / D_k^{vm_i}$ para todos os k's. O valor máximo de λ^i_{max} é limitado pelo recurso com maior valor de demanda de serviço. Por outro lado, o IDD pode se tornar o gargalo se a soma da utilização de CPU dedicada a cada VM atingir cap_0. Nesse contexto, definimos $U^*_{CPU,i}$ como a soma das utilizações de CPU no IDD devido a todas as VMs, exceto a vm_i, como:

$$U^*_{CPU,i} = \sum_{j=1}^{N} (D^{vm_0}_{CPU,j} \cdot \lambda^j) \quad \forall j \neq i \tag{8}$$

Note que, podemos assumir que $U^*_{CPU,i}$ é fixo, já que $D^{vm_0}_{CPU,i}$ e λ^i são parâmetros de entrada do modelo. Nesse contexto, a quantidade de recursos do IDD que vm_i pode utilizar é limitado por $cap_0 - U^*_{CPU,i}$. Consequentemente, λ^i_{max} pode ser obtida por:

$$\lambda^i_{max} \leq \min\left(\frac{cap_i}{max_{k=1}^{K} D_k^{vm_i}}, \frac{cap_0 - U^*_{CPU,i}}{D^{vm_0}_{CPU,i}}\right) \tag{9}$$

5. Experimentação

Esta seção apresenta resultados experimentais que guiaram a elaboração dos modelos discutidos na seção 4. Para todos os experimentos, utilizamos um servidor Intel Xen 64-bit com duas CPUs 3.2 GHz, com 2 GB de RAM, um disco com 7200 RPM e 8 MB de cache L2, e duas placas de rede Ethernet Gigabit Broadcom Realtek. Foi utilido o Xen versão 3.0.4. As VMs utilizam XenoLinux derivado de uma distribuição Debian Etch i386, kernel 2.6.16. A máquina cliente também utiliza essa mesma distribuição Linux. O servidor Linux é configurado com 1024 MB de RAM e cada VM é configurada com 512 MB de RAM bem como o IDD. Para experimentos com mais de duas VMs, cada uma utiliza 256 MB de RAM. O escalonador utilizado é o escalonador de créditos.

Para avaliarmos o desempenho de aplicações em ambientes virtuais, precisamos ser capazes de coletar medidas de desempenho no ambiente virtual. Como parte do nosso arcabouço de monitoramento, desenvolvemos uma aplicação, chamada *Xencpu*, para medir tempo ocupado de CPU no Xen. Esta ferramenta é baseada no código fonte da ferramenta *xm top*, que acompanha o código do Xen, e foi desenvolvida para permitir a

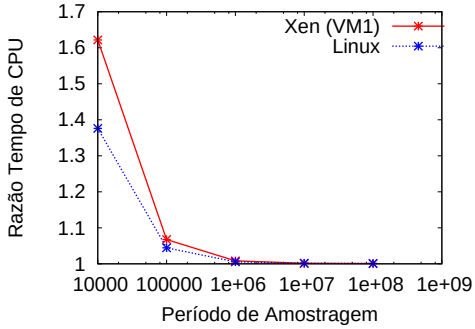


Figura 2. *Overhead* do Xenoprof e do Oprofile

execução automática de nossos scripts. O tempo ocupado de CPU no Linux e o número de pacotes nos dois ambientes foram obtidos do diretório */proc*. Para medirmos métricas como o número de instruções executadas e falhas (misses) nas caches do processador e na TLB[1], utilizamos o Xenoprof [13], uma ferramenta para o Xen que, periodicamente, coleta eventos de hardware do sistema. O código do Xenoprof é baseado em uma ferramenta do Linux chamada Oprofile [16] que possui a mesma funcionalidade do Xenoprof para máquinas executando Linux. Para medir as mesmas métricas no Linux, utilizamos o Oprofile.

Para realizarmos uma comparação justa de aplicações no Xen executando Xenoprof e no Linux executando Oprofile, definimos um período de amostragem no qual o Xenoprof e o Oprofile não causam um grande *overhead* ao sistema. A figura 2 compara o tempo de CPU para executar uma compilação de kernel (um dos nossos benchmarks) em uma máquina com Linux e na mesma máquina em uma VM do Xen. Para o Linux, plotamos valores obtidos executando o Oprofile relativo ao sistema sem executar o Oprofile. Para a VM, plotamos valores executando o Xenoprof relativos à VM executando o mesmo benchmark sem executar o Xenoprof. Podemos notar que o efeito de *trashing* é mais perceptível no Xen, já que existem mais instruções sendo executadas neste sistema devido ao custo da virtualização. Para períodos maiores de coleta, ambas as ferramentas não causam uma interferência significativa no desempenho do sistema. Para a compilação de kernel, escolhemos 10^7 como período de coleta para o evento *instruções executadas*. Foram realizados estudos similares para o experimento do servidor Web descrito a seguir e para cada evento analisado (além de instruções executadas, falhas nas caches e nas TLBs). A figura 2 é também útil para prover um entendimento do *overhead* do Xenoprof, já que este tipo de análise não foi apresentada anteriormente [13].

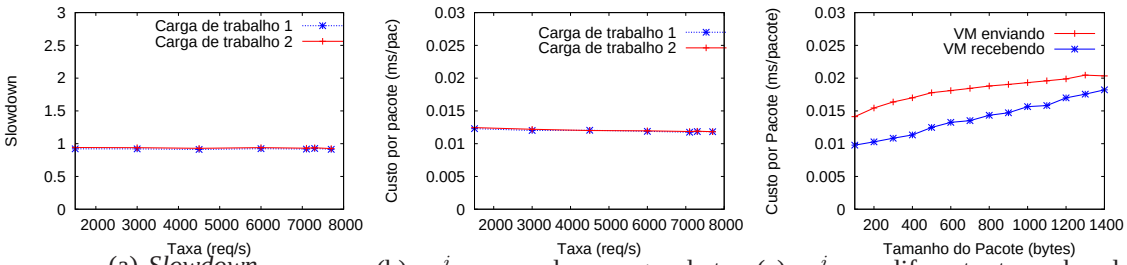
5.1. *Slowdown* para Classes de Aplicações

Para prover suporte experimental a nossa abordagem de criar um *slowdown* para representar o custo da virtualização para diferentes tipos de aplicações (seção 4.1), avaliamos o *slowdown* de um servidor Web que utiliza apenas conteúdo estático.

Utilizamos como clientes o httperf [14] e como servidor Web o Apache [1] versão 2.0.55. O httperf é uma ferramenta que permite gerar várias requisições HTTP e medir o desempenho do servidor do ponto de vista dos clientes. O httperf é executado na máquina cliente, enviando requisições ao servidor, medindo taxa de saída de requisições e tempo de resposta. Uma VM hospeda o servidor apache em uma única CPU e o IDD executa em outra CPU separadamente. As duas cargas de trabalho utilizadas pelos clientes e o conteúdo do servidor foram geradas pelo SPECWeb99 [18]. Essas cargas de trabalho, bem como o valor do *slowdown* obtido com elas, são utilizadas no nosso estudo de caso na seção 6. Ambas as cargas possuem distribuições e número de documentos acessados semelhantes. A figura 3(a) mostra o *slowdown* da CPU da VM para as duas cargas de trabalho como uma função da taxa de requisições. Note que os valores do *slowdown* para as duas cargas de trabalho são bastante próximos e não dependem da taxa de chegada de requisições. O *slowdown* médio para ambas as cargas é aproximadamente $0,92$.

Note que o *slowdown* não representa o valor total do *overhead* da virtualização, e consequentemente, pode ser menor do que 1 para uma aplicação em que parte de sua execução é feita pelo IDD (ex. aplicações que utilizam muitas operações de E/S). Como um exemplo, para uma de aplicação que faz uso intensivo somente da CPU como a

[1]TLB - Translation Lookaside Buffer



(a) *Slowdown* (b) cp^i para as duas cargas de trabalho analisadas (c) cp^i para diferentes tamanhos de pacotes

Figura 3. Análise de slowdown e custo por pacote

compilação de um kernel, o *slowdown* é de aproximadamente $1,03$. O *slowdown* depende das instruções que a VM precisa emular. Sendo assim, o *slowdown* pode apresentar valores diferentes para aplicações diferentes. Para uso prático, uma tabela de valores típicos de *slowdown* por classe de aplicação pode ser construída a partir de resultados experimentais obtidos a partir de benchmarks padrões.

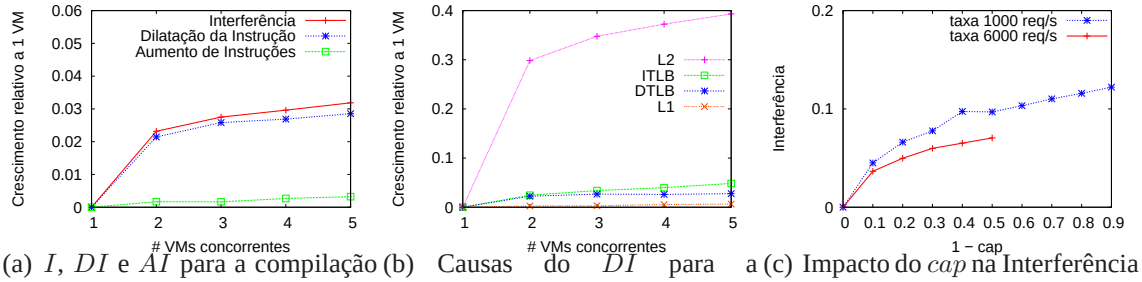
5.2. *Overhead* no IDD

Em seguida, provemos uma avaliação experimental para entendermos os principais fatores que envolvem a modelagem do *overhead* de E/S no Xen (seção 4.2). A configuração utilizada neste experimento é a mesma utilizada na análise do *slowdown*. A figura 3(b) mostra o custo de CPU no IDD por pacote da vm_i (cp^i) como uma função da taxa de requisições para as cargas utilizadas. Note que cp^i é constante, mesmo para grandes taxas de chegadas de requisições.

Para investigar os fatores que podem interferir no cp^i, analisamos o impacto no cp^i devido ao envio e o recebimento de pacotes de tamanhos diferentes. O netperf [15] foi utilizado para enviar tráfego de uma máquina real para uma VM no Xen. A VM executa em uma CPU e o IDD em outra. Além disso, observamos também uma VM enviando pacotes para um ambiente não virtual. O Netperf foi configurado para enviar pacotes a uma taxa fixa. Para simplificar os experimentos, utilizamos tráfego UDP.

A figure 3(c) mostra o cp^i como uma função do tamanho do pacote. Podemos ver que o cp^i para pacotes de chegada e de saída aumenta com o tamanho do pacote. Analisando o número de interrupções por pacote, entendemos que existe menos de uma interrupção por pacote quando pacotes são pequenos, diminuindo o cp^i. O componente principal que varia o número de interrupções por pacote é a taxa entre chegadas de pacotes. Sendo assim, uma premissa de nossos modelos é que a carga de trabalho utilizada para medir o número de pacotes por requisição da classe i (pr^i) e calcular $D^{vm_0}_{CPU,i}$ utilizando a equação 5, tenha as mesmas características da carga de trabalho (ex. distribuição do tamanho, tempo entre chegadas, etc) da aplicação alvo.

Considerando-se o tráfego gerado pela carga de trabalho 1, mostrada na figura 3(b), cp^i é uma função do número de pacotes que chegam e saem, variando entre 0,0117 e 0,0123 ms/pacotes. Sendo assim, assumimos um cp^i médio para a carga de trabalho 1 de 0.01195 ms/pacotes. Esse é o valor utilizado para calcular a utilização de CPU do IDD para a carga de trabalho 2, no estudo de caso da seção 6. Para E/S de disco, ainda não definimos uma métrica, já que a utilização de CPU do IDD devido a utilização de disco é muito pequena para as aplicações analisadas. Como trabalho futuro, planejamos estender os modelos para capturar a utilização do IDD devido à atividade de disco.



(a) I, DI e AI para a compilação do kernel (b) Causas do DI para a compilação do kernel (c) Impacto do cap na Interferência

Figura 4. Interferência e suas causas

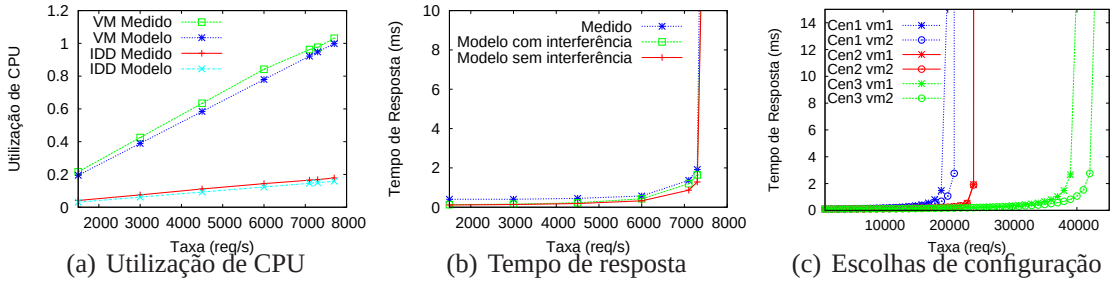
5.3. Interferência entre VMs

Esta seção analisa a interferência entre VMs quando elas compartilham a mesma CPU. Nosso objetivo neste trabalho é estudar as causas deste *overhead* extra e identificar as situações em que ele ocorre. Futuramente pretendemos estender nossos modelos para capturar explicitamente esse *overhead*.

Vamos começar decompondo os principais fatores que mudam quando aumentamos o número de VMs compartilhando a mesma CPU. A equação clássica de tempo de CPU depende de duas características (considerando a mesma plataforma de hardware): número de instruções executadas (IC) e o número de ciclos necessários para executar uma instrução (CPI). Chamamos de *aumento de instruções* (AI) e *dilatação da instrução* (DI) qualquer aumento no IC e na CPI, respectivamente, de uma VM devido a outras VMs executando concorrentemente na mesma plataforma de hardware. Como um exemplo, se o CPI médio de uma VM (vm_1) é 2.5 e, ao colocarmos uma outra VM na mesma CPU que vm_1, temos seu CPI aumentado para 2.6. Neste caso, o DI para vm_1 é 4%. De maneira semelhante à que definimos AI e DI, definimos *interferência* I como qualquer aumento no tempo de CPU de uma VM devido a outras VMs executando concorrentemente na mesma CPU.

Ambos, AI e DI são responsáveis pela interferência (I) quando múltiplas VMs compartilham a mesma CPU. Nesse contexto, nosso primeiro passo é quantificar I, AI e DI quando aumentamos o número de VMs em uma mesma CPU. Nosso experimento consiste em executar em cada VM uma aplicação de uso intensivo da CPU e de suas caches e observar uma destas VMs à medida que novas VMs são acrescentadas e executadas compartilhando a mesma CPU. Escolhemos como aplicação uma compilação do código do kernel 2.6.16 do Linux. A interferência é medida como o aumento no tempo de CPU consumido por uma VM para executar a aplicação à medida que vamos acrescentando novas VMs na CPU. AI e DI são medidos de forma semelhante a I. O IDD executa em uma CPU separada das outras VMs. A figura 4(a) mostra I, DI e AI como uma função do número de VMs. Podemos notar um grande aumento em I de 1 para 2 VMs e então, I começa a aumentar lentamente atingindo 3.19% para 5 VMs. O mesmo comportamento pode ser observado para DI, que é a principal causa da interferência para nossa aplicação. O aumento observado no AI é de apenas 0.32% para 5 VMs.

Como a principal causa da interferência é DI, vamos estudar as causas do DI. Analisamos quatro métricas: falhas nas caches de nível 1 (L1) e nível 2 (L2) do processador e também na TLB de dados (DTLB) e de instruções (ITLB), todas relativas a uma VM. Assumimos falhas na cache L1 como a soma de todas as falhas e acertos na cache L2, já que falhas na cache L1 é um evento que não pode ser obtido diretamente com o Xenoprof.



(a) Utilização de CPU (b) Tempo de resposta (c) Escolhas de configuração

Figura 5. Sumário da análise de desempenho do servidor Web

A figura 4(b) mostra estas métricas. Note que falhas na L2 possui um comportamento similar ao comportamento do DI, e consequentemente, para a aplicação utilizada, as falhas na L2 são as grandes responsáveis pela interferência entre VMs. Foram obtidos resultados semelhantes para o benchmark do servidor Web. Utilizando uma cópia de um arquivo de 2 GB como benchmark observamos um grande aumento nas falhas de ITLB (20%).

Por último, analisamos o impacto do parâmetro cap do escalonador de créditos na interferência. Este experimento é bastante similar ao apresentado anteriormente; entretanto, configuramos o ambiente virtual com o IDD em uma CPU e duas VMs compartilhando a outra CPU. A vm_1 executa uma sequência de compilações de kernel, causando interferência na vm_2, que executa um servidor Web e possui seu desempenho monitorado. O cap de ambas VMs é configurado de forma a sempre dividir todo o tempo de CPU de uma CPU para as duas VMs ($cap_1 + cap_2 = 1$). A figura 4(c) mostra, para duas taxas diferentes, I como uma função de $1 - cap$ da VM que executa o servidor Web. A interferência para as duas taxas de chegada de requisições aumenta à medida que o cap diminui e atinge 0.12 com 0.1 de cap para 1000 req/s. Para o outro experimento, não mostramos os valores da interferência para cap maiores do que 0.5 porque são pontos de saturação do servidor. A interferência para 6000 req/s atinge 0.07 para cap 0.5. Note que esses valores são altos e podem interferir em modelos de previsão de desempenho. Comparando os resultados para as duas taxas de chegada de requisições, notamos que para a taxa de chegada maior temos uma interferência menor. De fato, para pontos muito próximos do ponto de saturação, a interferência obteve valores menores que 1%. Um servidor Web ao receber uma taxa de requisições mais alta pode tirar maior proveito da localidade de caches do que recebendo uma taxa menor.

Baseado nos resultados desta seção, podemos ver que aplicações executando em uma mesma CPU podem introduzir um *overhead* considerável e difícil de se prever com modelos. No nosso modelo, a interferência pode ser introduzida como um fator multiplicativo do *slowdown* quando aplicações compartilham a mesma CPU. Entretanto, devido a complexidade de se prever a interferência, a maior parte dos modelos não aborda a questão da interferência [10, 17, 7]. Acreditamos, que esses resultados possam guiar o desenho de escalonadores para ambientes virtuais de forma a minimizar o uso compartilhado de CPU. Na próxima seção avaliamos o impacto de inflacionarmos ou não o *slowdown* pela interferência para um servidor Web virtualizado.

6. Estudo de Caso e Validação do Modelo

Nosso estudo de caso consiste em migrar um servidor Web de um sistema Linux para um ambiente virtual. A figura 6 sumariza as principais etapas para utilizar nosso arcabouço para previsão de desempenho. O primeiro passo consiste em determinar empiricamente o valor do *slowdown* e do cp^i para nossa aplicação. Para isso, utilizamos o

1: Criação da base de dados com:
- S_k^i *para* i, $1..N$ *e* k, $1..K$
- cp^i *e desempenho relativo de hardware para máquinas diferentes*

2: Medição das aplicações alvo no ambiente virtual, coletando:
- D_k^i *e* pr^i
- *Taxa de chegada de requisições* (λ^i)
- *Configuração de hardware para o ambiente não virtual*

3: Escolha da plataforma de hardware:
- *Definição de* N_{cpu}
- *Cálculo de* P_k^i *e ajuste de* cp^i *baseado em informações de hardware dos passos anteriores*

4: Escolha da classe de aplicações que melhor representa a aplicação alvo:
- *Definição de* S_k^i

5: Definição de cap^i, para i, $1..N_{CPU}$
6: Cálculo de $D_k^{vm_i}$, $U_k^{vm_i}$ e $R_k^{vm_i}$ para cada i e k

Figura 6. Passos para o uso dos modelos de previsão de desempenho

httperf enviando a carga de trabalho 1, primeiramente para o servidor Web executando no Linux, e depois para o servidor Web executando em uma VM que ocupa sozinha uma CPU (o IDD é configurado para usar a outra CPU). O *slowdown* e o cp^i foram medidos para diferentes taxas de chegadas de requisições, e são apresentados na figura 3(a) e 3(b). Os valores utilizados nos modelos correspondem às médias de todos os pontos destas figuras, que são 0,9193 e 0,01195, respectivamente. Para a validação do modelo utilizamos a carga de trabalho 2. Com base nesses números, medimos o desempenho da aplicação alvo no Linux (passo 2). A demanda de CPU (D_{CPU}^i) é 0,0706 ms/req. O número de pacotes durante o período observado é em média 1.723.020 e a taxa de requisição de entrada é o parâmetro que variamos em cada experimento. Como a plataforma de hardware é a mesma, P_k^i é 1 e N_{CPU} é 2 (os passos e 4 não são necessários, pois já escolhemos a aplicação). Configuramos o IDD executando somente em uma das CPU's e a outra CPU hospedando duas VMs. A vm_1 executa o servidor Web e a vm_2 executa uma compilação de kernel. A vm_2 é utilizada apenas para causar interferência na vm_1 e para demonstrarmos o uso do parâmetro cap. O valor de cap foi definido como 50% para cada VM e 100% para o IDD (passo 5). Baseado nestas informações, podemos computar as demandas no ambiente virtual utilizando as equações 2 e 5 e, então, medir a utilização de CPU para as VMs e para o IDD utilizando as equações 3 e 6, respectivamente (passo 6). A figura 5(a) mostra a utilização de CPU como uma função da taxa de chegada. Note que a utilização de CPU da VM é ajustada pelo cap, atingindo 100% quando a CPU real atinge 50% de utilização. Podemos notar que as utilizações previstas são muito próximas das medidas. Como a CPU da vm_1 é o ponto de contenção de desempenho, pela equação 9, $\lambda_{max}^1 = \frac{0.5}{D_{CPU}^{vm_1}} = 7.699$. Esse ponto de saturação pode ser observado na figura 5(b). Esta figura mostra o tempo de resposta médio das requisições. Para mostrar o impacto da interferência, plotamos nosso modelo contabilizando a interferência e sem contabilizar a interferência. A interferência que utilizamos para inflacionar o *slowdown* é a verdadeira interferência medida e foi obtida para cada ponto. Para os pontos em que o servidor Web está praticamente saturando, I é menor do que 1% e, consequentemente, ambas as curvas do modelo (com e sem interferência) saturaram em um ponto muito próximo. Ambos os resultados do modelo prevêem um ponto de saturação muito próximo ao ponto de saturação observado experimentalmente.

Para ilustrar a aplicabilidade do modelo, considere uma situação onde deseja-se saber a melhor configuração para implantar aplicações em uma determinada plataforma de hardware, mantendo a qualidade de serviço mesmo com altas de taxas de chegada. A

figura 5(c) mostra o tempo de resposta de duas aplicações Web, cada uma executando em uma VM separada, como uma função da taxa de chegada de requisições para três cenários diferentes. Para todos os cenários, configuramos a plataforma de hardware com *speedup* 2 e consideramos *slowdown* como 1 e interferência como 3%. A demanda de CPU do Linux para as aplicações 1 e 2 são 0.145 e 0.135 ms/req, respectivamente. Por simplicidade, assumimos que o tráfego de ambas aplicações causam a mesma demanda de CPU no IDD. O primeiro cenário consiste em utilizar uma máquina de 4 CPUs sendo cap como 1.5 para cada VM e 1 para o IDD. Como resultado ambas as VMs se tornam gargalos, atingindo cerca de 20.000 e 21.500 req/s, respectivamente. O segundo cenário, consiste em utilizar uma máquina com 8 CPUs com o mesmo *speedup* de hardware, aumentando o cap de cada VM para 3.5 e mantendo o cap do IDD igual a 1. Claramente, o IDD se torna o gargalo do sistema, o que causa que ambas as aplicações atinjam 24.000 req/s. O terceiro cenário aumenta o cap do IDD para 2 e reduz o cap de cada VM para 3. Isto aumenta a taxa de requisições máxima para ambas as aplicações. Esse exemplo mostra como modelos podem ser utilizados para identificar o gargalo do sistema e para definir a plataforma de hardware mais adequada para hospedar o ambiente virtual. Note que nossos modelos atuam na compra e definição do novo hardware. Por sua vez, o novo hardware pode ser gerenciado através de modelos de definição de capacidade auto-adaptativos [7].

7. Conclusões e Trabalhos Futuros

Este trabalho propõe e valida um arcabouço para prever o desempenho de aplicações no ambiente virtual Xen. Este é o primeiro trabalho que apresenta modelos para previsão de desempenho em ambientes virtuais e os valida com experimentação. Consequentemente, propomos um arcabouço para planejamento de capacidade bastante prático. Mais importante, planejamos construir uma ferramenta baseada no arcabouço proposto, capaz de prever o desempenho de aplicações em ambientes virtuais e determinar a melhor configuração e particionamento dos recursos de uma plataforma de hardware virtualizada.

A avaliação de desempenho apresentada mostra que o *overhead* de virtualização aumenta quando VMs compartilham a mesma CPU. Esse *overhead* extra não é contabilizado normalmente em modelos analíticos, nem considerado por escalonadores. Acreditamos que nossas descobertas possam guiar a criação de mecanismos capazes de diminuir o custo da virtualização e direcionar futuros modelos de desempenho.

Como trabalhos futuros, vamos validar nosso modelos para outras cargas de trabalho e desenvolver modelos para aplicações de classe fechada. Além disso vamos avaliar nosso arcabouço para diferentes ambientes virtuais, tais como VMWare e HPVM, e também para novo hardware da Intel para E/S em ambientes virtuais [8]. Além disso, pretendemos desenvolver uma ferramenta que utilize nosso modelo.

8. Agradecimentos

Este trabalho foi desenvolvido em colaboração com a HP Brasil P&D.

Referências

[1] Apache. `http://httpd.apache.org`.

[2] Y. Bard. Performance Analysis of Virtual Memory Time-Sharing Systems. *Proc. of IBM Systems Journal*, 14(4):366–384, 1975.

[3] Y. Bard. An analytic Model of the VM/370 System. *Proc. of IBM Journal of Research and Development*, 22(5):498–508, Set 1978.

[4] Y. Bard. The VM/370 Performance Predictor. *Proc. of ACM Computer Surveys*, 10(3):333–342, 1978.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *ACM SOSP*, Nova York, NY, Outubro 2003.

[6] F. Benevenuto, C. Teixeira, M. Caldas, V. Almeida, J. Almeida, J. R. Santos, and G. Janakiraman. Performance Models for Applications on Xen. In *Workshop on XEN in High-Performance Cluster and Grid Computing (XHPC'06)*, volume LNCS 4331, Sorrento, Itália, Dezembro 2006. Springer-Verlag.

[7] I. Cunha, J. Almeida, V. Almeida, and M. Santos. Self-Adaptive Capacity Management for Multi-Tier Virtualized Environments. In *IEEE/IFIP IM*, Munich, Germany, May 2007.

[8] I. V. T. for Directed I/O. `http://www.intel.com/technology/itj/2006/v10i3/2-io/5-platform-hardware-support.htm`.

[9] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe hardware access with the Xen virtual machine monitor. In *1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS)*, Washington, EUA, Outubro 2004.

[10] P. Garbacki and V. Naik. Efficient Resource Virtualization and Sharing Strategies for Heterogeneous Grid Environments. In *IEEE/IFIP IM)*, Munique, Alemanha, 2007.

[11] D. A. Menasce, L. W. Dowdy, and V. A. F. Almeida. *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall PTR, Upper Saddle River, NJ, EUA, 2004.

[12] D. Menascé. Virtualization: Concepts, Applications, and Performance. In *Proc. of CMG*, Orlando, FL, USA, Dezembro 2005.

[13] A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel. Diagnosing Performance Overheads in the Xen Virtual Machine Environment. In *ACM/USENIX VEE*, Chicago, IL, EUA, Junho 2005.

[14] D. Mosberger and T. Jin. httperf: A Tool for Measuring Web Server Performance. In *Proc. of WISP*, pages 59—67, Madison, WI, EUA, Junho 1998.

[15] Netperf. http://www.netperf.org/netperf/netperfpage.html/.

[16] Oprofile. `http://oprofile.sourceforge.net`.

[17] G. Rodosek, M. Gohner, M. Golling, and M. Kretzschmar. Towards an Accounting System for Multi-Provider Grid Environments. In *IEEE/IFIP IM*, Munique, Alemanha, 2007.

[18] SPEC. `http://www.spec.org`.

[19] A. P. technology. `http://enterprise.amd.com/us-en/AMD-Business.aspx`.

[20] I. V. technology. `http://www.intel.com/technology/computing/vptech/`.

[21] VMWare Web Site. http://www.vmware.com.

[22] A. Whitaker, M. Shaw, and S. Gribble. Scale and Performance in the Denali isolation kernel. In *Proc. of OSDI*, Boston, MA, EUA, Dezembro 2002.

[23] Xen Credit Scheduler. `http://wiki.xensource.com/xenwiki/CreditScheduler`.