Agregação de Interrupções de placas de rede para Infraestruturas em Nuvem

Eduardo Hideo Kuroda

Dissertação de mestrado apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciência da Computação

Programa: Mestrado em Ciência da Computação
Orientador: Prof. Dr. Daniel Macêdo Batista

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da Comissão Europeia e da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

São Paulo, Junho de 2013

Agregação de Interrupções de placas de rede para Infraestruturas em Nuvem

Esta é a versão original da dissertação elaborada pelo
candidato Eduardo Hideo Kuroda, tal como
submetida à Comissão Julgadora.

Resumo

As infraestruturas em nuvem se comportam com um desempenho mais baixo se comparadas com infraestruturas sem virtualização, em cenários onde há utilização intensiva da rede. Uma das principais causas é a arquitetura da virtualização de rede. Diferentemente da arquitetura de rede padrão, há alguns passos adicionais para transmitir e receber um pacote de informação dentro de uma máquina virtual o que implica em um custo adicional tanto na memória como no processamento. Para reduzir o custo do processamento, pode-se agregar várias interrupções geradas quando um pacote é enviado ou recebido. Isso reduziria a quantidade de processamento por pacotes, mas também aumentaria a latência. Essa estratégia é chamada agregação de interrupções. Nessa pesquisa criaremos um algoritmo para otimizar a agregação de interrupções com o objetivo de garantir requisitos de qualidade dos serviços de aplicações que fazem uso intensivo de rede em nuvens.
Palavras-chave: computação em nuvem, virtualização de rede, agregação de interrupções

Abstract

Cloud infrastructures shows inferior performance if compared to infrastructures without virtualization in scenarios where there is intensive use of network. One of the main reasons is the network virtualization's architecture. Differently from the standard network architecture, there are some extra steps to transmit and receive a packet of information inside of a virtual machine, which implies an additional cost in both memory and processing. To reduce processing cost, several interrupts can be coalesced when a packet is sent or received. This would reduce the amount of processing per packet, but also increase the latency. This strategy is called interrupt coalescence. In this research, we will create an algorithm to optimize the interrupt coalescence in order to ensure the requirements of quality of service of cloud applications that use network intensively.
Keywords: cloud computing, network virtualization, interrupt coalescence

Sumário

Lista de Abreviaturas

MV	máquina virtual (*virtual machine*)
CPD	centro de processamento de dados (*datacenter*)
CPU	unidade central de processamento (*central processing unit*)
E/S	entrada/saída (*In/Out*)
DMA	acesso direto a memória (*direct memory acess*)
IOMMU	unidade de gerenciamento de E/S da memória (*input/output memory management unit*)
dom0	domínio 0 (*domain zero*)
domU	domínio do usuário (*user domain*)
CDNA	acesso direto a memória concorrente (*concurrent direct network access*)
IRQ	pedido de interrupção (*interrupt request line*)
MTU	unidade máxima de transmissão (*maximum transmission unit*)
SR-IOV	virtualização de E/S de raiz única (*single root I/O virtualization*)
SaaS	software como um serviço (*software as a service*)
PaaS	plataforma como um serviço (*platform as a service*)
IaaS	infraestrutura como um serviço (*Infrastructure as a service*)
IP	protocolo de Internet (*Internet Protocol*)
QoS	qualidade de serviço (*quality of service*)

Lista de Símbolos

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Em infraestruturas de nuvem, recursos computacionais são controlados pelo fornecedor e alocados a qualquer momento de acordo com os requisitos do consumidor [GED+11]. Para alocar recursos a qualquer momento, a nuvem adota uma tecnologia chamada virtualização.

A virtualização divide um recurso com grande capacidade de processamento em recursos menores chamados de máquinas virtuais [BDF+03]. Com recursos menores, é possível fornecer ao consumidor uma quantidade menor de recursos que ainda satisfaçam seus requisitos e também alocar mais sob demanda [AFG+09]. Por exemplo, uma empresa de comércio eletrônico pode alugar uma quantidade x de processadores de uma máquina física na véspera do natal, afim de garantir que todos os clientes terão sucesso nas compras, e reduzir esse aluguel para $x/16$ após a data festiva, já que a quantidade esperada de acessos tende a cair. Apesar das máquinas virtuais ajudarem a aumentar a flexibilidade, elas ainda têm um desempenho abaixo da máquina pura quando executamos aplicações que usam muito a rede [CCW+08] [EF10] [Liu10] [WR12] [Rix08].

Uma das principais causas é o *hypervisor*, um módulo necessário para a gerência das máquinas virtuais, e sua arquitetura de virtualização de rede. Diferentemente da arquitetura de rede padrão, há alguns passos adicionais para transmitir e receber um pacote de informação que implicam em um custo adicional tanto na memória como no processamento [CCW+08] [EF10] [Liu10] [WR12] [Rix08].

Para reduzir o custo do processamento, uma proposta é adotar uma estratégia para gerenciar a agregação de interrupções [Sal07], [DXZL11]. Na agregação de interrupções, o *driver* da placa de rede não gera interrupções no sistema quando um pacote é recebido ou é transmitido com sucesso.

Ao invés disso, uma interrupção é gerada depois de um intervalo de tempo que um pacote é recebido ou transmitido, ou quando uma quantidade de pacotes é transmitida/recebida. Como a interrupção é gerada para um agrupamento de pacotes, ao invés de ser gerada para cada pacote, isso reduz a quantidade de interrupções por pacote, porém, pode atrasa o processamento de pacote já que cada pacote poderá ser processado somente quando a interrupção for gerada.

Embora focarmos em redes, qualquer dispositivo de E/S é capaz de gerar interrupções. Considerando outros dispositivos de E/S, as redes gigabits transferem com um taxa 10x maior que os discos rígidos e 1000x maior que os discos flexíveis como é possível ver na Figura 1.1 [Sta10].

A maioria dos trabalhos propostos em agregação de interrupções não analisam dispositivos de rede virtuais gerados pelo *hypervisor*. Estes participam diretamente do tráfego de rede em infraestruturas em nuvem. Em [DXZL11], foi proposto um algoritmo para dispositivo de redes virtuais, porém, foi considerado apenas a agregação de pacotes por intervalo de tempo. Essa pesquisa propõe algoritmos que automatizem a configuração de agregação de interrupções realizada pelo *hypervisor*. Será considerado tanto a agregação de pacotes por intervalo de tempo como também por quantidade de pacotes transmitidos ou recebidos. Os algoritmos tentam ajustar os parâmetro de agregação de interrupções dinamicamente de forma a garantir os requisitos de qualidade de serviço (*QoS* – do inglês *quality of service*) de aplicações em nuvem que façam uso intensivo dos enlaces de rede.

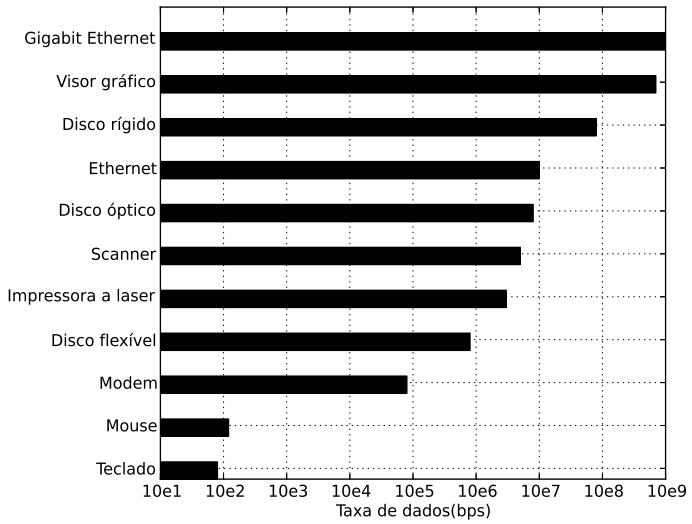

Figura 1.1: *taxa de transferência para diferentes dispositivos de E/S [Sta10]*

1.1 Objetivos

~~O objetivo dessa pesquisa é demostrar que os algoritmos de agregação de interrupções reduzem o uso da *CPU* por pacote na transmissão e recepção e mantém a latência consideravelmente baixa, dentro dos requisitos das aplicações.~~ O objetivo dessa pesquisa é demostrar que os algoritmos de agregação de interrupções propostos reduzem o uso da *CPU* por pacote na transmissão e recepção e mantém a latência consideravelmente baixa, dentro dos requisitos das aplicações.

1.2 Contribuições

As principais contribuições desse trabalho são:

- Fornecer um algoritmo capaz de garantir uma melhor utilização dos recursos de processamento e de comunicação na infraestrutura de nuvem para aplicações que façam uso intenso da rede.
- Automatizar as configurações de agregação de interrupções do *hypervisor* de acordo com a transmissão e recepção de pacotes.
- Desenvolver um módulo para simulações de nuvens que dê suporte a avaliação de desempenho de diferentes algoritmos de agregação de interrupções da placa de rede.

1.3 Organização do Trabalho

Este texto está organizado da seguinte forma: no Capítulo 2, são apresentados os conceitos de virtualização de servidores, virtualização de E/S, virtualização de rede, agregação de interrupções e computação em nuvem. No Capítulo 3 é apresentada uma revisão bibliográfica na área de virtualização de rede. Finalmente, no Capítulo 5 é apresentada a proposta de um algoritmo para a agregação das interrupções da placa de rede, de um mecanismo para automatização dos parâmetros do algoritmo e de um módulo para simulação de transferência de dados entre máquinas virtuais em nuvens que permite estudos relacionados com a gerência de interrupções das placas de rede.

Capítulo 2

Conceitos

O objetivo desse Capítulo é fornecer ao leitor o conhecimento necessário em agregação de interrupções, computação em nuvem e virtualização de rede. Ele está organizado da seguinte forma: a Seção 2.1 apresenta uma visão geral de computação em nuvem, a Seção 2.2 apresenta os conceitos em arquitetura de E/S em computadores, a Seção 2.3 explica a importância da virtualização na computação em nuvem e depois foca na virtualização em computadores, dispositivos de E/S e redes. As seções 2.4 e 2.5 explicam o funcionamento da estratégia de agregação de interrupções na recepção e transmissão de pacotes respectivamente. Por fim, a Seção 2.6 analisa como a aplicação das técnicas de agregação de interrupções afeta a virtualização de rede.

2.1 Computação em Nuvem

A computação em nuvem refere-se tanto a aplicações fornecidas como serviços por meio da Internet como também a sistemas de hardware e software dos CPDs (Centro de Processamento de Dados) que fornecem os serviços [AFG+09].

Como o termo computação em nuvem é muito abrangente, ele foi dividido em várias classificações [AFG+09], entre elas, o tipo de serviço o qual fornece. Seguindo essa classificação existem as nuvens que fornecem *software* como serviço (SaaS – *Software as a Service*), plataformas como serviço (PaaS – *Platform as a Service*), e infraestruturas como serviço (IaaS – *Infrastructure as a Service*). Nuvens que fornecem *SaaS* oferecem aplicações sob demanda para o cliente. Como exemplo temos o *Google Docs*[1] que fornece um *software* para edição de documentos como serviço. Nuvens que fornecem PaaS oferecem plataformas nas quais o cliente pode criar e implantar suas aplicações. O *Google App Engine*[2] e o *Windows Azure*[3] são exemplos desse tipo de nuvem. Nuvens que fornecem *IaaS* oferecem infraestruturas computacionais ao cliente. Um exemplo desse tipo de nuvem é o *Amazon EC2*[4] que fornece uma infraestrutura a qual emula um computador.

Nesse texto, iremos nos focar em fornecer infraestruturas, em específico, servidores como serviço. Cada serviço pode receber várias requisições para hospedar programas de desenvolvedores e, nesse caso, terá que implantá-los na infraestrutura. Quando um cliente, em algum momento, faz uma requisição para executar esse programa, a nuvem executa o programa internamente e repassa o resultado ao cliente. Para que isso seja possível, a infraestrutura de nuvem contém vários nós, os quais são recursos físicos, como computadores ou mesmo CPDs inteiros, que contêm e controlam várias máquinas virtuais (MVs) usando alguma técnica de virtualização. Cada requisição para implantar ou executar um programa é feita oferecendo as máquinas virtuais as quais estão contidas na infraestrutura.

[1] http://docs.google.com
[2] http://developers.google.com/appengine/
[3] http://www.windowsazure.com/en-us/
[4] http://aws.amazon.com/ec2/

2.2 Arquitetura de E/S em Computadores

Um computador, segundo o modelo de Von Neumann [Sta10], é formado por uma memória principal, uma unidade central de processamento e dispositivos de E/S como mostra a Figura 2.1. Cada dispositivo de E/S do computador é controlado por um módulo para E/S. Este módulo de E/S é necessário para que o processador possa se comunicar com um ou mais dispositivos de E/S. Os dispositivos de E/S possuem vários métodos de operação, diferentes formatos, comprimento de palavras e velocidade de transferência, o que faz cada módulo ter uma lógica específica para um dispositivo.

~~Quando o módulo oferece uma interface de alto nível do dispositivo ao processador, ele é chamado de canal de E/S. Já se o módulo oferece uma interface primitiva e requer um controle detalhado, ele é chamado de controlador de E/S. Os canais de E/S são normalmente usados em *mainframes*, computadores de grande porte, enquanto que controladores de E/S são usados por microcomputadores.~~

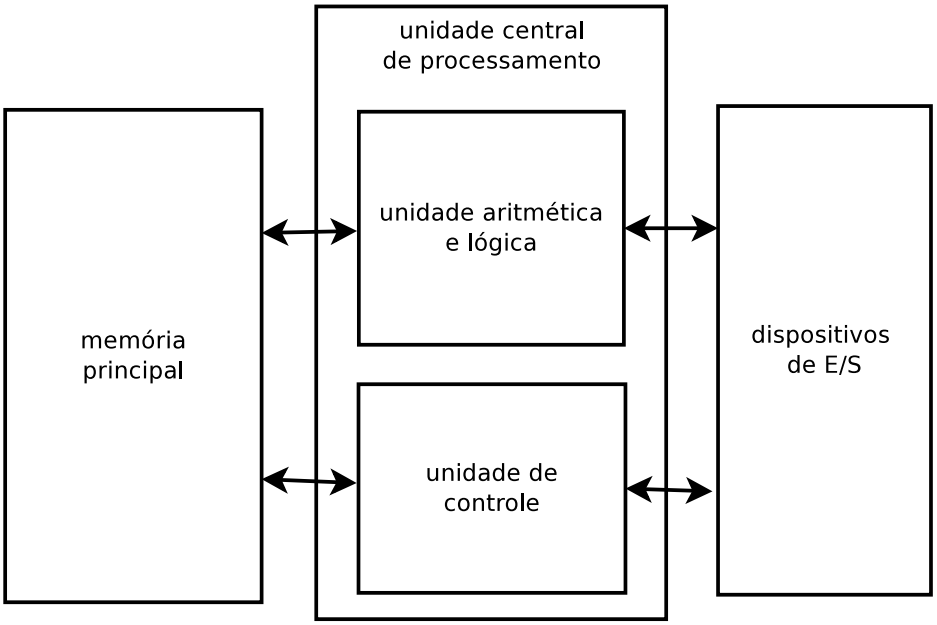

Figura 2.1: *Estrutura de um computador segundo o modelo de Von Neumann traduzida de [Sta10]*

Existem três técnicas possíveis para operações de E/S: E/S programada, interrupção dirigida à E/S e *DMA* (do inglês *direct memory access* – acesso direto a memória). Na E/S programada, dados são transferidos entre o processador e o módulo de E/S. O processador executa um programa e fornece a este, controle direto das operações de E/S. Um problema com essa estratégia é o intervalo de tempo que o processador precisa esperar para o dispositivo de E/S estar pronto para ser usado. Dentro desse intervalo, muitas instruções poderiam ser processadas. Na interrupção dirigida à E/S, um programa emite um comando de E/S e continua executando outras instruções. Ele é interrompido pelo módulo de E/S quando o último terminar seu trabalho. Como a interrupção dirigida à E/S não espera o dispositivo estar pronto, como na E/S programada, é possível processar uma quantidade de instruções maior que a última quando algum dispositivo de E/S é acessado. No *DMA*, um processador especializado em E/S recebe o controle das operações de E/S para mover um grande bloco de dados usando a memória principal. Nota-se que o processador não participa ativamente nessa técnica como nas anteriores, oque reduz o custo de processamento em relação as outras.

Na Figura 2.2, é possível observar o fluxogramas das três técnicas sendo aplicadas para receber um bloco de dados de um dispositivo de E/S. Tanto na E/S programada como na interrupção dirigida a E/S, percebe-se que o processador participa de todo processo, enquanto que o *DMA*, participa apenas na requisição de leitura e na recepção da interrupção do módulo de *DMA*, avisando

que o bloco de dados foi copiado. Atualmente, a técnica de E/S programada é pouco usada, pois é desperdiçado muito tempo de processamento e sempre podem existir aplicações que necessitam de processamento. Já a interrupção dirigida à E/S é normalmente usada para dispositivos de E/S que transferem quantidades pequenas de informação como o teclado e o *mouse*. Por fim, o *DMA* é normalmente usado para dispositivos que transferem tanto quantidades grandes como pequena de informação.

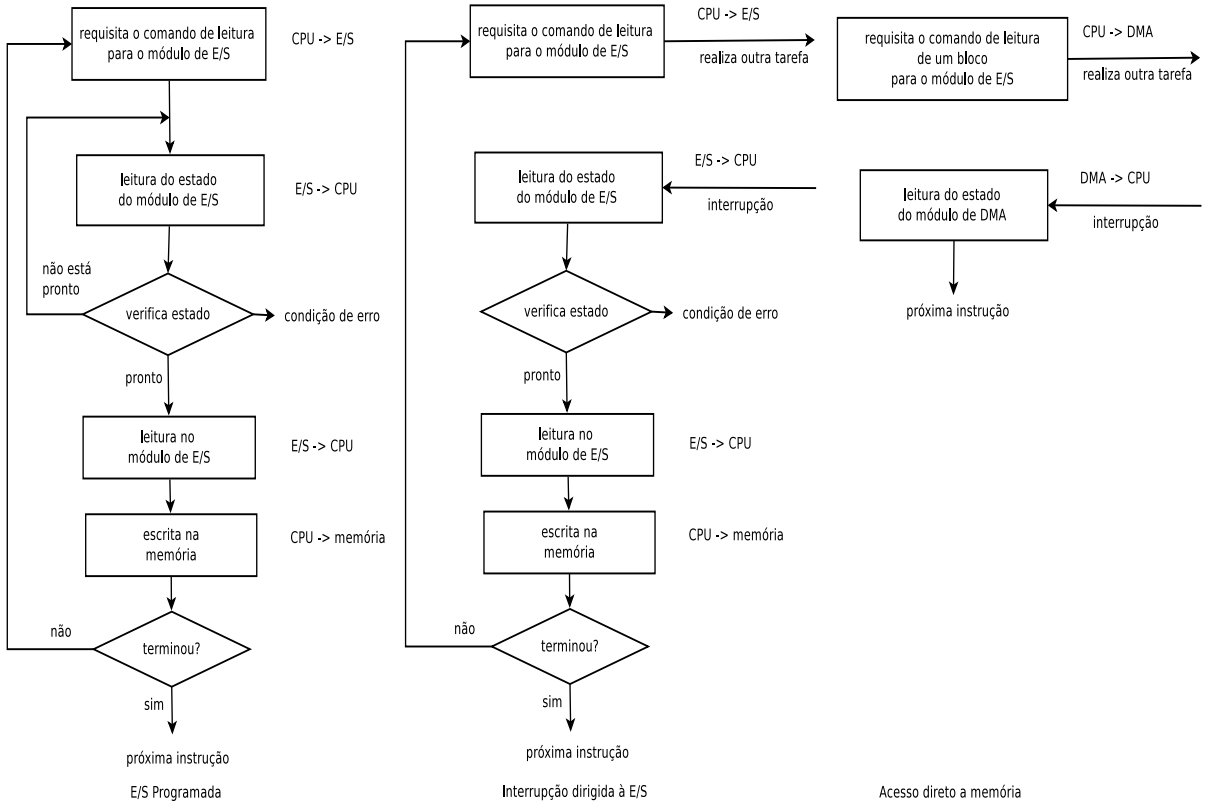

Figura 2.2: *três técnicas para operações de E/S traduzida de [Sta10]*

2.3 Virtualização

Na computação em nuvem, em particular quando se é fornecida uma infraestrutura para implantar aplicações (*IaaS*), a adoção da virtualização melhora a utilização dos recursos e protege o servidor de problemas que os software dos clientes possam causar em relação a servidores com máquinas puras [CCW+08]. Por exemplo, em um cenário com máquinas puras sem virtualização, um erro de programação que cause um laço infinito pode consumir toda a CPU do computador, atrapalhando todos os usuários daquela máquina. Em um cenário virtualizado, temos um isolamento entre os recursos das máquinas virtuais o qual impede uma máquina virtual de usar recursos de outra máquina. Assim, a única máquina afetada no cenário é aquela utilizada pelo programador. Além disso, não é possível expandir a quantidade dos recursos sem permissão do administrador da nuvem, dando mais segurança na infraestrutura em relação a infraestruturas sem virtualização. Além da segurança, outra consequência da virtualização, é o surgimento de um novo modelo de negócio chamado "pague somente quando usa", onde o cliente paga somente pelo tempo que o recurso é usado. Além disso, o cliente tem a impressão de estar utilizando um ambiente com recursos infinitos, já que podemos aumentar os recursos de uma máquina virtual sem interrupção do serviço e mais máquinas podem ser agregadas para prover o serviço [AFG+09].

Essas características beneficiam o lado do servidor, que não precisará fornecer um recurso físico inteiro para cada cliente e terá maior segurança e tolerância a falhas, já que cada sistema é

independente. Do lado do cliente, ele irá economizar dinheiro pelo novo modelo de negócio e terá recursos sob demanda.

Nos dispositivos de E/S, a virtualização permite a emulação de *hardware*. Em relação a flexibilidade, é possível mapear os dispositivos lógicos com as implementações físicas, garantindo uma maior portabilidade. Esse mapeamento pode também trazer novas funcionalidades ao recurso como: balanceamento da carga de trabalho e mascaramento das falhas.

2.3.1 Virtualização de Computadores

As nuvens normalmente são constituídas de CPDs que estão ligados de alguma forma por uma rede. A virtualização de servidores divide um computador, geralmente com grande capacidade de processamento, em recursos menores chamados de máquinas virtuais de modo que cada uma age como se fosse um computador separado, podendo ter inclusive, diferentes sistemas operacionais [BDF+03].

Com recursos menores, é possível fornecer ao consumidor uma quantidade menor de recursos computacionais que ainda satisfaçam seus requisitos e, caso ele necessite de mais recursos, é possível alocá-los sob demanda [AFG+09]. Segundo [CCW+08], as estratégias de virtualização podem ser divididas em 4 grandes categorias: virtualização completa, para-virtualização, virtualização em nível de sistema operacional e virtualização nativa.

Na virtualização completa também conhecida como emulação de hardware, um ou vários sistemas operacionais são executados dentro de um *hypervisor*. O *hypervisor*, chamado também de gerenciador de máquinas virtuais, fornece uma plataforma para os sistemas operacionais das máquinas virtuais e gerencia a execução delas.

No *hypervisor* da virtualização completa, é feita a interceptação, tradução e execução das instruções sob demanda dos sistemas operacionais das máquinas virtuais. Nessa estratégia, o núcleo do sistema operacional que roda o *hypervisor* não necessita de modificações. Dentro dessa categoria de *hypervisors* estão o *KVM*[1], o *XEN*[2], o *VMWare*[3] e o *VirtualBox*[4].

Diferentemente da virtualização completa, a para-virtualização exige uma modificação do núcleo para poder executar o *hypervisor*. Assim, caso não exista o código-fonte do sistema, não é possível usar essa estratégia. Na para-virtualização, o hardware virtual consegue conversar diretamente com o dispositivo emulado. Isso garante uma sobrecarga mínima em relação a tentar emular o dispositivo real. Nessa categoria estão incluídos o *XEN* e o *VMWare*.

A virtualização em nível de sistema operacional não tem um *hypervisor*. Ela modifica o núcleo do sistema isolando múltiplas instâncias do sistema operacional dentro de uma mesma máquina física. Nesse caso, como é feito apenas um isolamento entre as instâncias, estas ficam limitadas a usarem o mesmo sistema operacional. Está incluído nessa categoria o *OpenVZ*[5].

Por fim, a virtualização nativa é uma virtualização completa "melhorada". Ela aproveita o suporte de *hardware* para virtualização dentro do próprio processador. Isto permite que múltiplos sistemas operacionais rodem sobre outros, sendo capazes de cada um acessar diretamente o processador do hospedeiro. Como exemplos temos o *XEN*, o *VMWare* e o *VirtualBox*.

As virtualizações completa e nativa têm uma grande vantagem em relação às outras: não é necessário alterar o núcleo do sistema operacional da máquina hospedeira. Isto as tornam mais simples e mais portáveis já que sistemas operacionais com código fechados podem ser utilizados. A para-virtualização e a virtualização em nível de sistema operacional exigem uma modificação no núcleo da máquina hospedeira, porém, são as que tem um melhor desempenho pois elas têm acesso ao hardware físico. Comparando as duas, a virtualização em nível de sistema operacional é bem mais intrusiva e não permite a mudança do sistema operacional das máquinas virtuais, mas também tem um desempenho melhor que a para-virtualização [PZW+07] [CCW+08] [SBdSC] [CYSL10].

[1] http://www.linux-kvm.org/
[2] http://xen.org/
[3] http://www.vmware.com/
[4] http://www.virtualbox.org/
[5] http://wiki.openvz.org/

2.3.2 Virtualização de Dispositivos de E/S

Os dispositivos de E/S são intrumentos que recebem ou enviam dados para o sistema computacional como o *mouse*, o teclado e o monitor. Quando falamos em dispositivos físicos nos referimos ao dispositivo de E/S como *hardware*, enquanto que dispositivos lógicos se referem ao dispositivo em forma lógica. Com a virtualização de servidores, os dispositivos de E/S sofreram melhorias já que em um servidor não há apenas um único sistema operacional, mas sim, várias máquinas virtuais com um sistema dentro de cada uma.

Em [Rix08], foi separada a virtualização de E/S em duas categorias: privada ou compartilhada. Na virtualização de E/S privada, cada dispositivo físico é associado a apenas uma única MV enquanto que na virtualização de E/S compartilhada, o dispositivo é compartilhado por várias MV.

Comparando a virtualização de E/S privada com a compartilhada há uma subutilização na virtualização privada, pois enquanto uma MV não utiliza o dispositivo, outra poderia necessitar do seu uso. Por outro lado, o desempenho da virtualização compartilhada é pior já que divide o recurso com outras máquinas. Quando pensamos em aumentar o número de MVs, o custo da virtualização privada cresce absurdamente (com 10 MVs teríamos que ter 10 dispositivos físicos enquanto que na virtualização compartilhada, talvez até um dispositivo poderia ser o suficiente para resolver o problema).

Normalmente, a melhor opção é que o dispositivo físico seja compartilhado entre as máquinas, tanto pela possibilidade de escalar como pelo custo. Porém, disponibilizar de maneira compartilhada o acesso a dispositivos físicos pode trazer muitos problemas de segurança, dificultar o monitoramento das informações e a migração de máquinas virtuais [STJP08]. Problemas de segurança surgem porque o usuário de uma máquina virtual pode tentar acesso a uma outra máquina virtual justamente através do recurso que está sendo compartilhado [Rix08]. O monitoramento é dificultado porque certas ferramentas só fazem medições do dispositivo físico e como ele está associado a várias máquinas virtuais, fica difícil separar as informações específicas de cada uma dentro do agregado [GED+11]. Já a migração é dificultada porque uma máquina virtual só poderá ser migrada para uma máquina física que possua o mesmo *hardware* onde a máquina virtual está executando (a migração permite que uma máquina virtual seja movida de um recurso físico para outro de forma transparente para o usuário, sem perda de conectividade)[AU09].

Para contornar esse problema, normalmente, na virtualização em nível de sistema operacional, o núcleo do sistema gerencia a utilização do dispositivos entre as máquinas virtuais como mostra a Figura 2.3. Já em *hypervisors* como *XEN*, *KVM* e *VMWare*, como cada máquina possui seu próprio núcleo, é muito complexo fazer o conjunto de núcleos gerenciar o dispositivo entre as máquinas. Uma possível solução seria restringir o acesso ao dispositivo físico para apenas uma máquina virtual e o acesso a esse dispositivo pelas outras máquinas virtuais é feito através dessa máquina. Como é possível ver na Figura 2.4, a máquina virtual 0 gerencia o dispositivo de E/S enquanto que as máquinas virtuais 1 e 2 acessam o dispositivo se comunicando com o núcleo do sistema 0. Essa restrição traz uma perda de desempenho em relação a ambientes que não usam virtualização quando o uso da rede é intensa, porém, garante segurança já que é possível monitorar o trafego de todas as máquinas através da máquina que gerencia o dispositivo, por exemplo [CCW+08] [EF10] [Liu10].

Em [WR12], foram feitas algumas menções sobre o uso de técnicas de virtualização de E/S que desacoplam o dispositivo físico da sua implementação lógica. Dentre as vantagens, ele cita a melhor utilização dos recursos e a economia de custos em relação a sistemas que estão com o dispositivo físico acoplado com a sua implementação lógica, pois vários sistemas podem aproveitar o mesmo recurso. Em relação a flexibilidade, é possível mapear os dispositivos físicos com as implementações lógicas, garantindo uma maior portabilidade. Esse mapeamento pode também trazer novas funcionalidades ao recurso como: balanceamento da carga de trabalho e mascaramento das falhas. A funcionalidade de suspender, migrar e continuar uma máquina virtual também é possível, pois com o dispositivo físico desacoplado da implementação lógica, é possível reconectar a máquina virtual em outra máquina física com uma configuração diferente. Outra funcionalidade trazida com a virtualização é a interposição e transformação das requisições virtuais de E/S. Isso permite que as

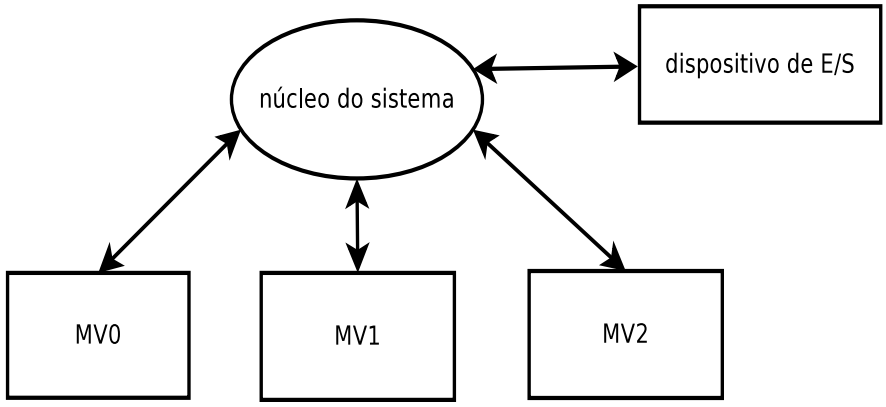

Figura 2.3: *compartilhamento de um dispositivo de E/S na virtualização em nível de sistema operacional*

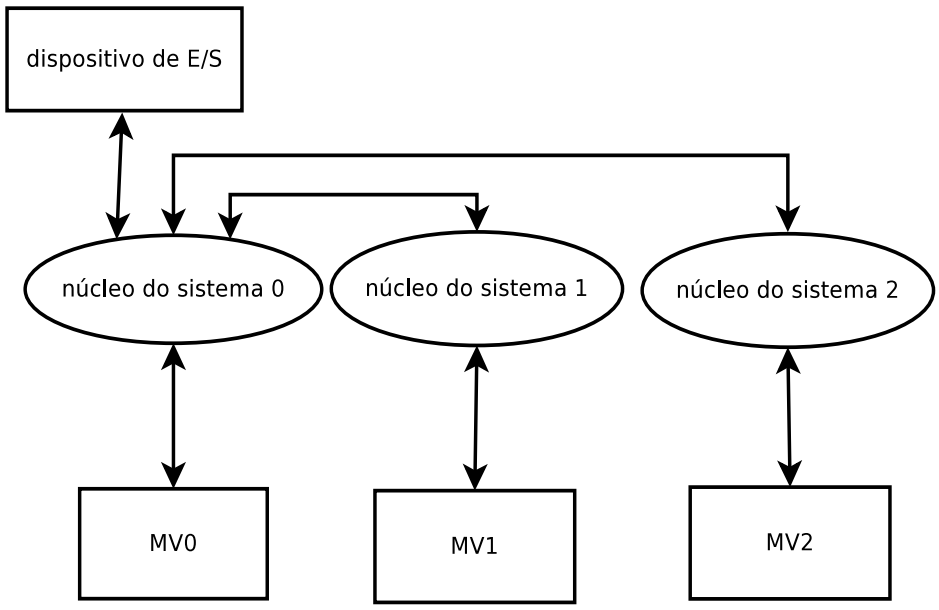

Figura 2.4: *compartilhamento de um dispositivo de E/S na virtualização que utiliza hypervisors*

requisições que passam pelo dispositivo lógico sejam transformadas. Em um exemplo de leitura e escrita no disco, além de simplesmente ler/escrever no disco, torna-se possível guardar uma cópia da informação antiga como cópia de segurança. Outra possibilidade é criptografar a informação quando alguém escrever no disco, dificultando outras pessoas de acessarem o seu conteúdo escrito.

2.3.3 Virtualização da Rede

A virtualização de rede, que também é um dispositivo de E/S, tem algumas particularidades em relação a outros dispositivos. Segundo os autores em [Rix08], a complexidade de virtualizar a rede é muito maior pelo fato de muitas vezes não se conhecer o destino de uma informação, pois esse está fora do sistema,diferente por exemplo do disco rígido que só se comunica com o sistema. Outra dificuldade é necessidade de estar preparado a qualquer momento para receber e responder ao tráfego da rede, diferente da virtualização de disco em que a leitura e escrita só ocorre quando requisitada pela máquina virtual.

Virtualização da Rede no XEN

O *XEN* é um *hypervisor* de código aberto disponível para arquiteturas de máquina física x86, x86_64, IA64, ARM. Ele permite a virtualização nativa, completa e para-virtualizada de sistemas operacionais *Windows*, *Linux*, *Solaris* e diversos outros sistemas baseados no BSD [Spe10].

No *XEN*, o *dom0* ou domínio zero é a primeira máquina virtual iniciada. Ela tem certos privilégios que as outras máquinas virtuais não têm, como iniciar novas máquinas e acessar o hardware diretamente. Os `domUs` ou domínios do usuário são máquinas virtuais que, por padrão, não tem alguns privilégios que o `dom0` tem como o acesso direto ao *hardware*. Assim, é necessário um mecanismo para conseguir acessar o dispositivo de rede [Spe10].

No *XEN*, para todas as máquinas conseguirem acessar o dispositivo de rede ao mesmo tempo, existem dois tipos de configuração: ponte e roteador. Ambas as configurações seguem os conceitos dos equipamentos de mesmo nome que existem na interconexão de redes de computadores. Todos os dois tipos encaminham pacotes entre domínios baseados nas informações que os próprios pacotes contêm, porém a ponte se fundamenta nos dados da camada de enlace enquanto que o roteador se fundamenta nos dados da camada de rede [BM99]. Podendo trafegar pacotes entre domínios, os `domUs` conseguem enviar e receber pacotes do dispositivo de rede com o `dom0` como intermediário.

Em [Eas07], foi descrita a implementação da configuração de ponte na qual uma ponte virtual (`xenbr0`) é criada dentro do `dom0` como é possível ver na Figura 2.5. Essa ponte está ligada na interface de rede física `peth0`. A interface `vif0.0` está sendo usada para tráfegos de/para `dom0` e as interfaces `vifX.0`, onde X é um valor maior que 0, estão sendo usadas para tráfegos de/para algum `domU`. Como é possível observar, todo pacote que é recebido ou transmitido para alguma máquina virtual tem que passa pela ponte dentro do `dom0`. A configuração de roteador é muito semelhante à configuração da ponte, porém, ao invés de existir uma ponte virtual, o *dom0* possui um roteador virtual que é configurado para encaminhar pacotes *IP* entre os domínios e os *domUs*.~~levantam uma interface virtual `vifX.0` para se comunicar com o *dom0*. A figura ?? mostra um exemplo de configuração do tipo roteador.~~

Em [Jam04], foi feito um experimento comparando a ponte virtual e o roteador virtual. Os resultados foram semelhantes tanto na largura de banda como na latência e no uso do processador. Nessa pesquisa focaremos na configuração de ponte devido ao fato dessa configuração trabalhar numa camada de rede mais baixa e por outros trabalhos relacionados [CCW+08], [EF10], [WR12], [SBB+07], [STJP08], [ON09], [Cor09], [LGBK08], [AMN06], [JSJK11], [FA12], [DXZL11] terem feito experimentos com essa configuração.

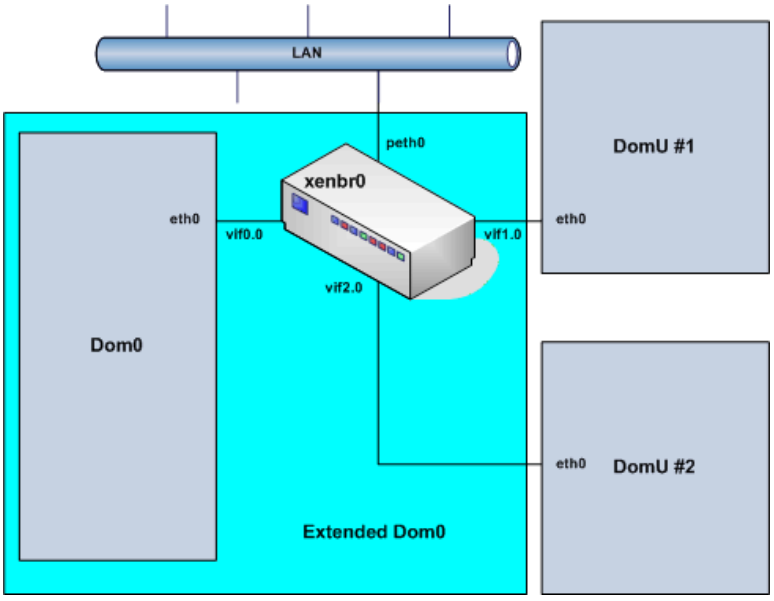

Figura 2.5: *ponte virtual criada no XEN [Eas07]*

Na Figura 2.6 vemos a arquitetura da virtualização da rede usando ponte no *XEN* segundo [STJP08]. Para transmitir/receber um pacote no `domU` é usado o canal de E/S (*I/O channel*). Esse canal evita que cada pacote tenha que ser copiado de um domínio a outro. Para tal, o `domU` compartilha algumas páginas de sua memória e informa a referência delas por esse canal para o outro domínio mapeá-las em seu espaço de endereço. Quando algum domínio envia algum pacote para essas páginas, uma notificação é enviada para o outro domínio.

O canal de E/S consiste de notificações de evento e um *buffer* de descrição em anel. A notificação de evento avisa que algum usuário do domínio deseja enviar informações. O *buffer* de descrição em anel guarda os detalhes de requisições entre o *driver* de *frontend* (*netfront*) que fica no interior do *dom0* e o *driver* de *backend* (*netback*) que fica dentro de um `domU`. ~~O domínio que controla os *drivers* (domínio do *driver*) por padrão é o *dom0*. Porém, em alguns casos o *driver* pode sobrecarregar o processamento do *dom0*, então, às vezes, ele é separado em um domínio exclusivo.~~

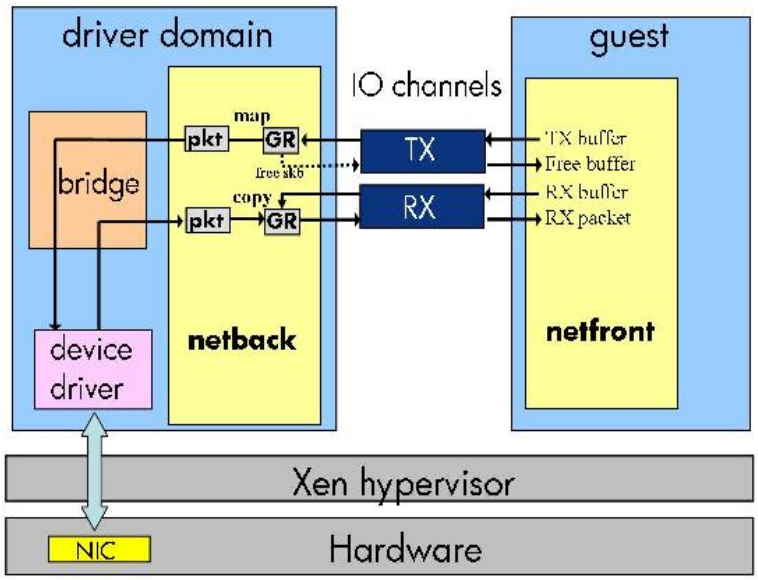

Figura 2.6: *arquitetura da rede virtual no XEN [STJP08]*

Para o *dom0* ter acesso às páginas da memória do *domU* é necessário um mecanismo de permissão. Neste, o *domU* fornece páginas vazias da sua memória para serem usadas como *buffer* de E/S. Essas páginas são passadas como referência na descrição da requisição.

Na transmissão de pacotes, o `domU` coloca o pacote no *buffer* de E/S, as referências de suas páginas de memória no *buffer* de descrição e notifica o *dom0* através de uma interrupção. O *dom0* por sua vez, lê o *buffer* de descrição, mapeia as páginas recebidas no seu espaço de endereços e pede para transmiti-las através da ponte. Quando o dispositivo físico confirmar a transmissão, o `domU` libera as páginas do *buffer* de E/S.

Na recepção, o *netfront* informa as possíveis páginas da memória que podem ser usadas como *buffer* de E/S ao *netback*. Quando algum pacote chega pelo dispositivo físico, este envia uma interrupção de chegada de pacote à ponte dentro do *dom0*. A ponte então avisa o *netback* correto sobre a chegada de pacotes. O `netback` o copia para uma página da memória que foi fornecida pelo *netfront* e envia uma interrupção para o mesmo. Quando o `netfront` recebe a interrupção, ele pega o conteúdo que está no *buffer*, envia para o seu sistema e libera as páginas fornecidas. A arquitetura de virtualização de rede do XEN apresentada nesta subseção é utilizada por outros *hypervisors* como, por exemplo, o KVM e o VMWare [STJP08].

2.4 Agregação de Interrupções na Recepção

Quando o tráfego de pacotes possui uma taxa de transmissão da ordem de Gbps no meio físico, a quantidade de interrupções devido a chegada de pacotes é muito grande podendo sobrecarregar o processamento [DXZL11]. Isso ocorre porque as interrupções têm prioridade absoluta sobre todas as outras tarefas e se a taxa de interrupções é suficientemente elevada, o sistema gastará todo seu tempo para respondê-la e o rendimento do sistema cairá para zero. [Sal07].

A agregação de interrupções é uma das propostas da literatura para resolver esse problema [Sal07]. Ela pode ser feita através de um conjunto de parâmetros do *driver* de redes se este o suportar. O objetivo é reduzir a quantidade de interrupções na transmissão/recepção de pacotes dentro de um intervalo de tempo ou número de pacotes em troca de aumentar a latência da rede.

Para isso é possível manipular 4 parâmetros: `tx-frames`,`rx-frames`,`tx-usecs`,`rx-usecs` (a descrição de cada parâmetro está na tabela 2.1). Como pode-se notar na tabela 2.1, a agregação de interrupções depende do tamanho do *buffer* de transmissão e recepção. O *buffer* pode ser tanto um espaço de memória da máquina (*DMA*) como uma memória interna da placa de rede. Caso este seja pequeno, vários pacotes serão descartados durante o tráfego de pacotes por falta de espaço, caso seja grande, pode aumentar a latência por ter muitos pacotes esperando serem lidos dentro dele.

O NAPI (New API) [CRKH05] é uma interface que permite utilizar técnicas de agregação de interrupções para dispositivos de rede no núcleo do Linux. O objetivo dela é reduzir a carga extra do processamento na recepção de pacotes de vários dispositivos. Para isso, no momento em que há uma grande quantidade de tráfego em vários dispositivos de rede, ao invés do *driver* gerar uma interrupção para cada pacote que recebe, o núcleo desabilita as interrupções e passa a checar continuamente a chegada de pacotes em cada dispositivo. Caso o sistema não dê conta de manipular os pacotes, ele passa a descartá-los antes de levá-los ao núcleo. Esse processo é chamado de *polling*. Como nem sempre se tem um tráfego grande de pacotes, usar essa estratégia o tempo todo pode gerar um atraso considerável na rede. Assim, o modo de interrupção por pacote padrão e o modo de *polling* ficam se alternando de acordo com o tráfego. O controle de quando o sistema deve entra ou sair no modo de *polling* e quantos pacotes ele deve aguardar por interrupção em cada dispositivo de rede são definidos por um parâmetro chamado "peso". Com pesos altos, a quantidade de pacotes esperada para gerar uma interrupção ou para entrar em *polling* no dispositivo é maior, enquanto que com pesos baixos, a quantidade de pacotes esperada é menor. [cor05].

Tabela 2.1: *Parâmetros para agregação de interrupções*

nome	descrição
`tx-frame` N	gera uma interrupção quando a quantidade de pacotes transmitida chegar a N
`rx-frame` N	gera uma interrupção quando a quantidade de pacotes dentro do buffer de recepção chegar a N
`tx-usecs` N	gera uma interrupção N microssegundos depois que um pacote for transmitido
`rx-usecs` N	gera uma interrupção N microssegundos depois que um pacote for recebido

2.5 Agregação de Interrupções na Transmissão

Tanto a transmissão quanto a recepção de pacotes podem gerar interrupções com uma frequência grande [MCZ06]. A transmissão gera uma interrupção quando um pacote é transmitido com sucesso e a recepção gera uma interrupção quando um pacote é recebido [CRKH05]. A diferença entre elas

é que enquanto a transmissão pode controlar os pacotes que são enviados pelo sistema, a recepção não consegue controlar os pacotes que chegam. Assim, na transmissão podemos reduzir de outras formas a quantidade de interrupções. Uma das principais propostas da literatura é o *GSO* [Cor09]. O *GSO* (*Generic segmentation offload*) permite ao *driver* de rede segmentar os pacotes, uma tarefa que normalmente é feita pelo processador.

Atualmente, o tamanho do pacote é limitado pela *MTU*. No protocolo *Ethernet* ela tem como valor padrão 1500 *bytes*. Esse valor acabou sendo adotado na época do crescimento da Internet pelos limites de *hardware* da época e infelizmente continua até hoje. Assim, não é possível enviar pacotes maiores que 1500 *bytes* pela Internet, o que força o sistema operacional a segmentar seus dados em pacotes pequenos para conseguir enviá-los. Isso sobrecarrega o processador tanto para segmentar os dados, como para enviar e receber esses pacotes.

Com a segmentação sendo feita apenas no momento da transmissão dos pacotes pelo *GSO*, pode-se configurar o *MTU* da interface de rede do sistema acima do limite do dispositivo físico. Com um *MTU* maior, o pacote é segmentado em pedaços grandes e em menor quantidade quando o sistema manda transmiti-lo. Com menos pacotes, a quantidade de interrupções por pacote é reduzida. Na recepção, o *LRO* (large receive offload) e o *GRO* (generic receive offload) [Cor09] são soluções baseadas no *GSO*, onde os pacotes são montados quando recebidos. O *LRO* monta cada pacote agregando os pacotes *TCP* que chegam, porém, se por exemplo, existir uma diferença nos cabeçalhos do pacote *TCP*, haverá perdas na montagem, pois o pacote será montado sem considerar essa diferença. Já o *GRO*, restringe a montagem dos pacotes pelos cabeçalhos, o que não gera perdas e, além disso, o *GRO* não é limitado ao protocolo *TCP*. Apesar da proposta permitir a montagem de pacotes, como já foi dito, não é possível controlar a chegada de pacotes, o que força a adoção de alguma técnica de agregação de interrupções como o *NAPI* para conseguir montar os pacotes.

2.6 Agregação e Virtualização de Rede

No contexto da virtualização de rede, como foi possível observar na arquitetura da virtualização da rede no *XEN*, muitos passos extras são feitos durante recepção e transmissão de pacotes, fazendo aumentar o número de interrupções.

Na virtualização de rede do *XEN*, dois *drivers* virtuais (*frontend*,*backend*) são criados pelo próprio *XEN* para ligar o `dom0` com um `domU`.

A estratégia de agregação, então, pode ser feita tanto no *driver* físico como no *driver* virtual de *frontend* e *backend*.

Em [DXZL11], foi proposta uma otimização por agregação de interrupções na recepção dentro dos *drivers* virtuais. Os autores perceberam que o pacote passa por duas camadas de *drivers* virtuais de rede antes de chegar no destino. O primeiro é o *driver* de *backend* que fica na ponte e o outro é o *driver* de *frontend* que está dentro da máquina virtual. Considerando estas duas camadas, a combinação de agregação de interrupções nas duas causaria um atraso adicional na recepção. Nessa pesquisa eles focaram em otimizar os *drivers* virtuais, deixando de fora o *driver* físico e analisaram apenas o intervalo para gerar as interrupções e não a quantidade de pacotes para gerar as interrupções.

Mesclar as interrupções em cada dispositivo tem certas diferenças que devem ser consideradas.

2.6.1 Driver do Dispositivo

A agregação no *driver* do dispositivo físico é complexa, uma vez que afeta o tráfego de pacotes em todas máquinas virtuais e, consequentemente, em todas as aplicações que usam a rede. Também necessita que a placa de rede tenha suporte a agregação. Quando modificamos o *driver* físico, é possível termos problemas com os requisitos das aplicações.

Como exemplo, podemos ter duas aplicações, onde uma requer uma baixa latência e baixa largura de banda, e a outra requer muito processamento e alta largura de banda. Se não agregarmos as interrupções, a primeira aplicação funcionará bem, pois nenhum pacote precisa esperar para

ser enviado, enquanto que a segunda funcionará mal, porque a rede irá precisar de muito processamento e a aplicação também. Se agregarmos, a primeira funcionará mal, pois a agregação irá provocar um atraso considerável na rede, e a segunda funcionará bem, porque a agregação reduziu o processamento da rede liberando processamento para a aplicação.

Uma possível solução para conseguir satisfazer os requisitos seria forçar todas as aplicações da infraestrutura a terem os mesmos requisitos realocando as máquinas com requisitos de aplicações diferentes para outras infraestruturas.

2.6.2 Driver de Backend

A agregação no *driver de backend*, diferente do *driver* do dispositivo físico afeta apenas as aplicações de uma determinada máquina virtual. Uma vez que o *driver de backend* está no domínio do administrador, consumindo processamento junto com vários outros *drivers de backend*, reduzir suas interrupções aliviaria o processamento da rede por máquina virtual do domínio do *driver* e poderia permitir que mais máquinas virtuais usassem a rede.

Seria necessário analisar os requisitos de aplicação de cada máquina virtual para definir os parâmetros da agregação de cada *driver de backend*. Pelo fato do *driver de backend* e *driver de frontend* serem virtuais e desacoplados da lógica do *driver* de rede físico, eles estariam aptos a usar técnicas de agregação independente do *driver* de rede físico.

2.6.3 Driver de Frontend

A agregação no *driver de frontend* depende somente das aplicações da máquina controladora do *driver de frontend*. O ganho pode ser menor em relação ao *driver de backend* já que irá reduzir a interrupção no núcleo de uma máquina virtual que é isolada das outras.

Capítulo 3

Revisão Bibliográfica

3.1 Objetivo

O objetivo dessa revisão foi analisar e estudar as maneiras já existentes de otimizar a utilização dos recursos da rede em infraestruturas de máquinas virtuais utilizadas para a criação de nuvens e os problemas em aberto. Cada estratégia sugerida pode atender bem a um cenário, porém, em outros casos, essa mesma estratégia pode ser pouco eficiente devido a dinamicidade da rede e os diferentes requisitos de *QoS* de um usuário.

3.2 Critérios de Seleção

Para seleção das referências, em cada artigo encontrado foi lido o seu resumo e classificado manualmente em três categorias de acordo com sua relevância: alta, média, baixa.

Os artigos de relevância alta foram lidos por completo e resumidos. Os artigos de relevância média tiveram a leitura de sua introdução e a mudança da sua relevância para baixa ou alta. Por fim os artigos com relevância baixa não foram lidos. A tabela 3.1 mostra os artigos coletados e sua relevância.

Tabela 3.1: *Artigos selecionados e suas relevâncias*

referência	relevância
[CCW+08]	alta
[EF10]	alta
[WR12]	alta
[Liu10]	alta
[Rix08]	alta
[SBB+07]	média-baixa
[STJP08]	alta
[ON09]	alta
[Cor09]	alta
[LGBK08]	média-alta
[AMN06]	alta
[JSJK11]	alta
[FA12]	alta
[DXZL11]	alta
[SEB05]	alta
[Sal07]	alta
[int07]	alta

3.2.1 Resumo Sintetizado

Em [CCW+08], o autor fez uma comparação entre o *XEN*, *VMWare* e *OpenVZ*. A partir dos experimentos foi concluído que o *hypervisor XEN* tem um desempenho baixo em termos de ~~atraso na rede~~ latência, porém alto em termos de largura de banda em relação a um ambiente com *OpenVZ* e um ambiente sem virtualização, enquanto que o *OpenVZ* tem uma perda em largura de banda, mas um atraso pequeno. Quanto ao *VMWare*, ele teve um desempenho baixo tanto em atraso quanto em largura de banda. Os autores não entram em detalhes sobre os motivos dos resultados terem sido esses.

Em [EF10], foi estudado a relação entre o número de núcleos e o número de MVs usando *XEN* e *Eucalyptus* como infraestrutura de nuvem. Foi concluído que a virtualização funciona bem para aplicações que não se comunicam muito, enquanto que em aplicações que são sensíveis a latência, houve uma perda de desempenho em relação a um ambiente não virtualizado. Outra conclusão foi que quanto maior o número de máquinas virtuais, maior a sobrecarga na *CPU*. A explicação para isso, segundo o autor, está na forma como é implementada a virtualização da rede. O hardware físico só pode ser controlado por um sistema (`dom0`), enquanto que os outros (`domUs`) para conseguirem fazer alguma operação de E/S pela rede, devem passar por esse sistema através de um canal. Isso forma um gargalo no `dom0`.

Em [Rix08], foi feita uma revisão sobre a virtualização de rede. No texto o autor cita que a virtualização de rede impacta diretamente no número de servidores que podem ser diretamente consolidados dentro de uma única máquina física. Porém, as técnicas modernas de virtualização têm gargalos significantes, o que limita o desempenho da rede. Ele sugere um ganho de desempenho fazendo o dispositivo ter a capacidade de ler e escrever diretamente na memória da MV ao invés do processador da máquina virtual gerar interrupções cada vez que alguma informação entra ou sai pelo dispositivo. Essa funcionalidade é chamada acesso direto a memória (*DMA*). Apesar disso, o dispositivo pode escrever em uma posição da memória que não pertence a MV, podendo assim, causar problemas em outros processos da máquina física. Assim, foi criado a unidade de gerenciamento de E/S da memória (*IOMMU*). No *IOMMU* a memória é restrita para o dispositivo de acordo com a máquina virtual que controla esse dispositivo. Atualmente os *hypervisors* modernos, como o *XEN*, utilizam essas técnicas [BDF+03] .

Como atualmente um processador possui vários núcleos, pode-se aproveitar esses núcleos para criar multi-filas nas interfaces de rede. O autor cita que pesquisadores do laboratório da HP e *Citrix* eliminaram a ponte no domínio de E/S para associar as máquinas virtuais diretamente com o *driver* de *backend* através das multi-filas, evitando a necessidade de sincronização das mensagens e multiplexação/demultiplexação da rede. Como benefícios do uso da multi-fila se teve: a redução da carga extra na fila e a eliminação de cópias entre o domínio de E/S e a máquina virtual, pois, a multiplexação não é feita. Por outro lado, é necessário que cada informação seja enviada para a fila correta e que a *CPU* consiga aguentar a carga extra gerada pelas múltiplas filas.

Ainda em [Rix08], na arquitetura de virtualização de rede CDNA (*acesso direto a memória concorrente*) foi empregada a proposta de multi-filas, em adição foi removido o domínio de E/S. Sem o responsável por controlar as filas, o *hypervisor* passa a considerar cada conjunto de fila como uma interface de rede física e associa o controlador a uma MV. Assim, cada MV consegue enviar ou receber informações diretamente da rede sem nenhuma intervenção do domínio de E/S. Como consequência, a carga extra é reduzida pelo número reduzido de interrupções (antes era necessário interromper tanto o domínio de E/S como as MVs em cada transmissão/recepção). Pela MV poder acessar diretamente a interface de rede, ela também pode acessar algum local indevido da memória por *DMA*. Para contornar esse problema o autor sugeriu o uso de *IOMMU*.

Em [WR12], são citados diversos desafios e problemas na área de virtualização de E/S: a carga extra no *hypervisor*, a complexidade em gerenciar recursos (escalonamento e prioridades) e a dificuldade de dar uma semântica ao hardware virtual.

Em [Liu10], foram feitos diversos experimentos com virtualização de E/S baseados em *software* (*virtio*) e em hardware (*SR-IOV*) usando o *hypervisor KVM*. O *virtio* é um padrão do Linux para *drivers* de rede e disco que estão rodando em um ambiente virtual cooperado com um *hypervisor*. Apesar de diferentes, ele tem o mesmo padrão arquitetural que a virtualização de rede do *XEN*. Já o *SR-IOV* é uma especificação que permite a dispositivos *pci-express* fornecerem interfaces extras com funcionalidades reduzidas para serem usadas pelas máquinas virtuais diretamente.

Foram analisadas diversas métricas: a largura de banda, a latência e uso do processador. Na latência, o *virtio* teve um desempenho muito baixo. A explicação, provada desabilitando a função de agregação de interrupções na transmissão, é que o hospedeiro atrasa o envio do pacotes para ser enviado em rajadas, mas mesmo assim, seu desempenho sem mitigação ainda perdeu próximo de 20 microssegundo em relação a máquina não virtualizada. Quando a opção de agregação é desabilitada, isso provoca uma perda de desempenho pois cada pacote que é transmitido gera uma carga de trabalho no *CPU*. Com a mitigação a carga por pacote é reduzida. Já o *SR-IOV* (single root I/O virtualization) teve um desempenho próximo da máquina pura perdendo apenas alguns microssegundos devido a virtualização da interrupção.

Na largura de banda, a transmissão em todas as configurações pareceu ter o mesmo desempenho. Já na recepção o *SR-IOV* se aproximou da máquina pura, mas o uso da sua *CPU* foi muito maior que as demais. No *virtio*, ele não conseguiu um bom desempenho, mas o uso de sua *CPU* foi baixa. No experimento de uso da memória na recepção, o *SR-IOV* teve um uso muito menor que o *virtio*, assim, o autor concluiu que o mal uso da largura de banda na recepção do *virtio* foi pelo uso excessivo da memória, o que explica também o baixo uso da *CPU*.

Em [STJP08], foi proposto modificar a arquitetura do *driver* de E/S do *XEN* para conseguir melhorar o uso da *CPU*. Dentro dos problemas que o autor encontrou está o excesso de cópias de dados. Para reduzir o excesso de cópias, foram propostos otimizações nas operações de remapeamento de páginas tanto na transmissão como na recepção. No *hypervisor*, ele conseguiu uma economia de 56% no uso do processador.

Em [ON09], foi analisado o desempenho de um sistema virtualizado com *XEN* aplicando a estratégia *LRO (large receive offload)* onde ainda dentro do *driver* da placa de rede são recebidos e reunidos os pacotes de informações que tiveram que ser segmentados. Nesse experimento foram medidos a vazão da rede variando o tamanho da mensagem e o tamanho da *MTU* (unidade máxima de transmissão). Os resultados mostraram um ganho de 8% a 14% na vazão da rede.

Em [LGBK08], foram propostas duas otimizações na virtualização de rede: o escalonamento ciente do cache do processador e o roubo de créditos de escalonamento para a recepção de pacotes. A primeira ideia é fazer com que o `domU` e o `dom0` passem a compartilhar o cache, assim, a comunicação entre domínios é reduzida. A segunda otimização foca priorizar a recepção de pacotes onde o uso do processador é alto. Nos experimentos comparando a estrutura padrão de virtualização e a estrutura modificada com as otimizações, foi apresentado um aumento de 96% na largura de banda.

Em [AMN06], foram pesquisadas as principais causas de carga extra na virtualização de E/S. No experimento eles estudaram dois modos de virtualização de E/S: o `domU` e o `dom0` na mesma *CPU* e em *CPUs* distintas. O resultado mostrou que nos dois métodos, tanto a transmissão como a recepção de pacotes apresentaram uma perda de desempenho de mais de 50% quando comparado com a máquina física. Também foi notado que rodar o `domU` e o `dom0` em *CPUs* distintas é mais custoso que rodar elas juntas na mesma *CPU*.

Em [JSJK11], foi estudado sacrificar o isolamento que existe entre as máquinas virtuais para conseguir reduzir a carga extra do processador. Os resultados mostram uma redução de 29% no uso do processador e 8% de ganho de largura de banda na transmissão de pacotes grandes.

Em [FA12], foram feitos experimentos em torno do problema da carga extra na virtualização da rede. Para isso, os autores propuseram adequar o balanceamento de interrupções para demostrar a possibilidade de reduzir o número de pacotes perdidos. O resultado foi que um balanceamento adequado pode melhorar muito o desempenho, porém, o comportamento é difícil de ser previsto, dificultando a elaboração de um algoritmo. Uma proposta futura sugerida foi deixar o núcleo do sistema automatizar o processo de balanço e analisar os resultados. Quando aparecerem bons resultados, congelar a configuração de interrupção.

Eles também, discutiram a possibilidade de usar a função de agregação existentes nos *drivers* das placas de rede modernas para um trabalho futuro.

Em [DXZL11], foram propostas otimizações para reduzir a carga extra na virtualização da rede. Uma das otimizações foi agregar eficientemente as interrupções virtuais e a outra escalar o lado da recepção. A agregação de interrupção normalmente é usada quando a transferência de pacotes do meio físico é muito alta. Com uma transferência grande, a placa de rede passa a trabalhar intensamente sobrecarregando o processador com interrupções. Na virtualização da rede, a transferência de pacotes passa a gerar interrupções físicas e virtuais.

A agregação de interrupções virtuais pode ser feita no *driver* virtual de *backend* (na ponte dentro do `dom0`) ou no *driver* virtual de *frontend* (dentro de um `domU`).

A ideia de escalar o lado da recepção foi baseado na ideia de paralelizar o *driver* virtual de *backend* tentando aproveitar melhor as propriedades de um processador multinúcleo. Assim, foi introduzido o conceito de *RSS* que balanceia a carga de trabalho eficientemente entre os processadores.

O resultado no experimento de agregação de interrupções foi um ganho de até 76% na largura de banda em relação a configuração padrão e no experimento de escalar o lado da recepção a largura de banda foi 2,2 vezes a largura de banda da configuração padrão.

Evaluating System Performance in Gigabit Networks [SEB05]

Em [SEB05], é feita uma análise e simulação sobre o impacto da sobrecarga de interrupções no desempenho do sistema operacional em redes de alta velocidade. O principal problema que eles exploraram é a grande quantidade de interrupções gerada na recepção de pacotes. Como a interrupção tem prioridade máxima em relação a outras tarefas, ela acaba consumindo todo tempo de processamento, impedindo outras tarefas de serem realizadas e, consequentemente, reduzindo a taxa de transferência do sistema a 0. Essa situação é conhecida como *"livelock"*.

Na Figura 3.1, é mostrado um gráfico de carga do sistema por taxa de transferência. Na curva ideal, conforme a carga do sistema aumenta, a taxa de transferência passa a aumentar proporcionalmente, ou seja, quanto maior a velocidade de chegada dos pacotes, maior a quantidade de pacotes processada. Porém, como praticamente todo sistema tem uma capacidade finita de processamento, ele não recebe e processa pacotes além de sua velocidade máxima. Essa velocidade é chamada de *MLFRR* (do inglês *Maximum Loss-Free Receive Rate* – máxima velocidade de recepção livre de perdas). Na curva aceitável, quando o sistema chega à *MLFRR*, ele passa a não ter ganho na taxa de transferência pelo limite de processamento, e depois disso, a curva se comporta praticamente como uma constante. Por outro lado, se a rede for sobrecarregada na entrada, as interrupções geradas na chegada dos pacotes irão impedir que o pacote seja processado e a taxa de transferência do sistema cairá para 0 como é possível ver na curva *livelock*.

Para analisar a situação de sobrecarga de interrupções, os autores modelaram o sistema como uma fila M/M/1/B com chegada de pacotes em *Poisson* de velocidade λ e média efetiva de tempo de serviço de $1/\mu$ que tem uma distribuição geral (os autores não justificam o motivo de terem modelado segundo essa distribuição). O sistema pode usar ou não *DMA*. Sem *DMA*, o processador gerencia a recepção de pacotes. Quando o processador é interrompido enquanto está processando um pacote pela chegada de um outro pacote, o tempo para processar é estendido para realizar uma cópia individual do outro pacote que chegou para a memória do sistema. Com *DMA*, a placa de

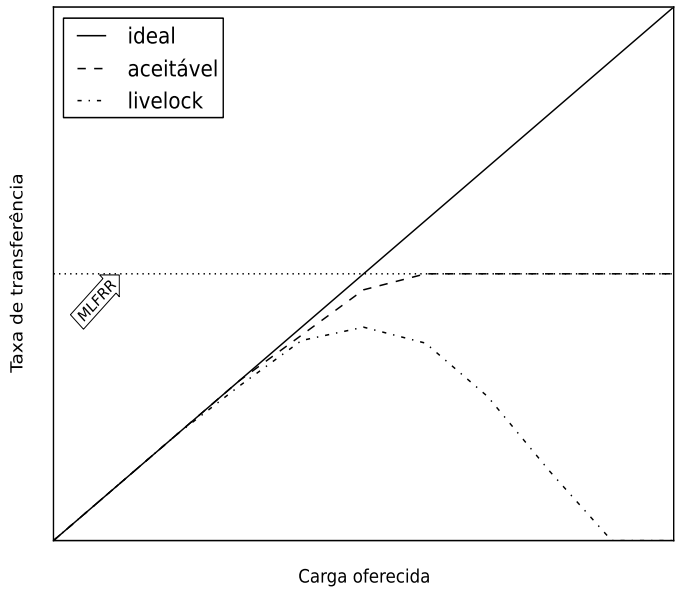

Figura 3.1: *Livelock na recepção de pacotes [SEB05]*

rede tem acesso direto a memória. Quando um ou mais pacotes chegam enquanto que o sistema está ainda processando um outro pacote, todos são processados sem estender o tempo de processar.

Foram feitos experimentos analisando o sistema ideal, *DMA* habilitado e *DMA* desabilitado. Com pouco tráfego de pacotes, a taxa de transferência de todos foi a mesma. Já com muito tráfego, a taxa de transferência do sistema com o *DMA* habilitado teve uma queda menor na taxa de transferência que o sistema sem o *DMA* habilitado e o sistema ideal se apresentou com taxa de transferência constante.

To Coalesce or Not To Coalesce [Sal07]

Em [Sal07], continuação do artigo [SEB05], é analisado o desempenho de duas técnicas de agregação de interrupções: baseada em contagem e baseada em tempo. Na técnica baseada em tempo, o *driver* da placa de rede não gera interrupções no sistema quando um pacote é recebido. Ao invés disso, uma interrupção é gerada depois de um intervalo de tempo que um pacote é recebido. Já na técnica baseada em contagem, é gerada uma interrupção quando uma quantidade de pacotes é recebida.

A conclusão tirada nos modelos analíticos é que a agregação funciona melhor que o modelo de interrupção comum quando se tem um grande tráfego na rede. Porém, para um tráfego pequeno, a interrupção comum superou a agregação. Os autores sugerem monitorar o tráfego e fazer a troca entre a interrupção comum e a agregação de interrupções. Eles também citam um momento que pode ser usado para indicar a condição de sobrecarga. Este momento pode ser usado para a troca. Outras importantes conclusões são que na agregação, são necessários valores altos de parâmetros em tráfegos intensivos, que para tráfegos tolerantes a latência, o uso de agregação é interessante independente do tráfego e que para tráfegos de tempo não-real é interessante usar a agregação baseada em tempo ao invés da baseada em contagem por impor um limite de atraso na agregação.

Interrupt Moderation Using Intel GbE Controllers [int07]

Em [int07], é citado o problema da grande quantidade de interrupções gerada na transmissão e recepção de pacotes. Para resolvê-lo, os autores propuseram o uso de temporizadores internos da placa de rede para moderar a quantidade de interrupções geradas. Os temporizadores são divididos

em temporizador absoluto, pacote e mestre. O temporizador absoluto inicia uma contagem regressiva quando o primeiro pacote chega ou é enviado. No momento que a contagem chega a zero, é gerada uma interrupção no sistema. Este temporizador é eficiente quando se tem muito tráfego de pacotes, pois muitos pacotes chegam/são enviados até o temporizador gerar a interrupção, reduzindo a quantidade de interrupções por pacote. Por outro lado, ele não é eficiente quando há pouco tráfego porque poucos pacotes chegam/são enviados até o temporizador gerar a interrupção e por atrasar as interrupções e, consequentemente, as informações que devem chegar ao sistema.

O temporizador de pacotes também inicia uma contagem regressiva quando o primeiro pacote chega ou é enviado e também gera uma interrupção quando a contagem chega a zero, mas ele é reiniciado sempre que um novo pacote chega. Isso reduz a latência quando há pouco tráfego no enlace, pois a interrupção é gerada quando o temporizador percebe que nenhum pacote será mais enviado/recebido, mas quando há muito tráfego, ele pode nunca gerar a interrupção, pois o temporizador estará sempre sendo reinicializado pelos pacotes que chegam/são enviados. O temporizador mestre é usado para otimizar os outros temporizadores. O temporizador absoluto e o temporizador de pacotes podem ser combinados para chegar a um bom resultado.

Além dos temporizadores já citados, existe um outro mecanismo chamado limitador de interrupções. Esse mecanismo também é um temporizador de contagem regressiva e limita o número de interrupções por segundo. Quando o temporizador inicia a contagem regressiva, este também começa a contar o número de interrupções que foi gerado. Quando a contagem chega a zero, o contador de interrupções também é zerado. Se o número de interrupções ultrapassar o limite estabelecido, as interrupções geradas são adiadas até o contador ser reinicializado.

Um algoritmo foi proposto para moderação de interrupções que ajusta dinamicamente o valor do limitador de interrupções. Dependendo do padrão de E/S, é usado um valor no limitador. O padrão de E/S é categorizado em: baixíssima latência, onde o tráfego é mínimo e predomina os pacotes pequenos, baixa latência, onde o tráfego também é minimo e há um significante percentual de pacotes pequenos, e intermediário, onde há muito tráfego de pacotes medianos. Não foi possível entender como os autores chegaram a esses valores e porque eles resolveram dividir o padrão de E/S dessa forma.

3.2.2 Resumo Conclusivo

Nessa revisão, foram encontrados diversos artigos com propostas que modificam diferentes partes da infraestrutura: *driver* de rede, placa de rede física, arquitetura da virtualização da rede e núcleo do sistema operacional. Essa variação dificultou um pouco a correlação entre os artigos. Em [int07], foi proposto um algoritmo para a moderação de interrupções de acordo com o padrão de E/S. Porém, não ficou claro como os autores criaram o algoritmo. Em [SEB05], [Sal07] foram criados modelos analiticos e simuladores para analisar a sobrecarga de interrupções. Entretanto, eles não avaliaram um cenário onde existem dispositivos de E/S virtualizados.

Nos experimentos, percebe-se que a maioria usou o *hypervisor XEN* e alguma distribuição *Linux* como sistema operacional. Uma possível explicação é que ambos têm código aberto e portabilidade.

Todos que propuseram alguma estratégia as validaram através de medições em infraestruturas reais. Uma possível causa seria a facilidade e o baixo custo em montar uma infraestrutura com virtualização e controlar todo o processo do experimento.

A revisão ajudou a entender melhor a área de virtualização de rede. Diversas formas de melhorar o desempenho foram encontradas. Porém, não foi encontrada nenhuma proposta de um algoritmo para agregar as interrupções que seja independente das tecnologias de nível mais baixo.

Capítulo 4

Motivação

4.1 NAPI

A *NAPI* (*New API*) [CRKH05] é um conjunto de interfaces oferecido pelo núcleo do Linux que os *drivers* dos dispositivos de rede usam para agregar interrupções. O objetivo dela é reduzir a carga extra do processamento na recepção de uma grande quantidade de pacotes em um ou mais dispositivos de rede. Para isso, no momento que uma grande quantidade de pacotes for enviada para o dispositivo de rede, ao invés do dispositivo enviar uma interrupção ao *driver* para cada pacote que chega, o *driver* desabilita as interrupções na chegada do primeiro pacote e processa continuamente próximos pacotes.

No processo dos pacotes, o *driver* envia uma tarefa de recepção de pacotes na fila de *polling* do núcleo do sistema[cor05]. Cada tarefa da fila de *polling* é processada pelo núcleo do sistema e a quantidade de pacotes que essa tarefa poderá processar é definida por uma variável peso. Quanto maior o peso, mais pacotes poderão ser processados, mas não existe uma relação clara entre os pacotes e o peso, pois cada *driver* faz uma implementação diferente da recepção de pacotes. Durante o processo de recepção, é verificada a quantidade de pacotes recebida, se uma estimada quantidade não for recebida, a tarefa é retirada da fila de *polling*, a interrupção é reativada e será necessária a chegada de outro pacote para a tarefa retornar a fila, já se essa quantidade é processada, a tarefa é encerrada e recolocada na fila [cor05].

Comparando a *NAPI* com a estratégia comum de interrupção, a *NAPI* têm como vantagem a quantidade reduzida de interrupções geradas pelo dispositivo já que a interrupção é desabilitada durante o processo de pacotes, isso influência diretamente no desempenho da largura de banda de tráfegos intensivos.

4.2 Experimento

Para analisar o desempenho da *NAPI* em dispositivos de rede virtuais, foram feitos experimentos variando o parâmetro peso do *driver* de rede e1000 da *Intel*. Esse *driver* implementa *NAPI* e tem o código-fonte claro e bem escrito, sendo usado por vários *hypervisors* como *XEN*, *VirtualBox*, *VMWare* e *KVM* para processar a recepção e envio de dados pela rede na máquina virtual. Na sua implementação de recepção de pacotes, o peso define a quantidade limite de pacotes que a tarefa de recepção poderá coletar. Se a quantidade de pacotes atingir esse limite, a tarefa é recolocada na fila de *polling*, caso contrário, ela é removida da fila.

Como *hypervisor* foi usado o *VirtualBox* por sua instalação ser rápida e sua interface ser simples e clara. A máquina física contém um processador i7-2620M de dois núcleos e quatro fluxos de execução, 8 *Gigabytes* de memória *RAM* e sistema operacional *Mac OS X* 10.6.8. A máquina virtual usa dois fluxos de execução, 5 *Gigabytes* de memória *RAM* e sistema operacional *Ubuntu* 11.10 com núcleo *Linux* 3.0.43.

O desempenho foi analisado verificando a largura de banda, o processamento da máquina física e virtual, a quantidade de interrupções gerada pela recepção e transmissão de pacotes e a quantidade

de pacotes processada por ciclo de *pooling*. A banda foi medida usando o programa `iperf` com protocolo *TCP* e *UDP* durante 30 segundos, o processamento usando o programa `top`, a quantidade de interrupções usando `itop` e a quantidade de pacotes por ciclo de *pooling* através do `dmesg`, um comando do *Linux* para imprimir as mensagens do núcleo de sistema na saída padrão. Como a quantidade total de pacotes processada durante o experimento pode ultrapassar 1 milhão, o *buffer* pode encher rapidamente e, consequentemente, descartar informações. Assim, foi necessário aumentar o comprimento do *buffer* de mensagens do núcleo alterando o parâmetro `LOG_BUF_LEN` do núcleo do sistema.

Em todos os experimentos, o processamento da máquina virtual chegou ao seu limite de processamento sendo o gargalo nos experimentos. Já no processamento da máquina física os experimentos não passaram de 50% de uso.

Na Figura 4.1, é mostrada a largura de banda processada para diferentes valores de pesos e protocolos. Com protocolo *TCP*, comparando o peso com valor igual a 2 e a 64, a largura de banda é maior com 64, pois cada tarefa de recepção processa mais pacotes, e assim, necessita de executar menos tarefas de recepção para a mesma quantidade de pacotes com peso igual a 2. Percebe-se que a quantidade de interrupções por segundo, na Figura 4.2, foi praticamente a mesma com peso igual a 2 e 64.

Na Figura 4.3, vemos frequência de pacotes processados por ciclo de *polling* com peso igual a 2 e protocolo TCP, nota-se que o sistema processa muitas vezes 2 ou nenhum pacote. Observando os dados de quantidade de pacotes processada no ciclo de polling com peso igual a 2, percebemos uma alternância nos valores 0 e 2 o que dificulta o sistema se manter em *polling*. Esse comportamento é devido ao *TCP* que obriga o sistema a responder ao remetente do pacote para receber o próximo pacote, e assim, atrasar o processo de envio. Nesse cenário, o sistema não consegue entrar em *polling* por um período longo pois o processador atinge seu limite antes. Na Figura 4.4, vemos que o sistema dificilmente se manteria em *polling* pois raramente processa 64 pacotes.

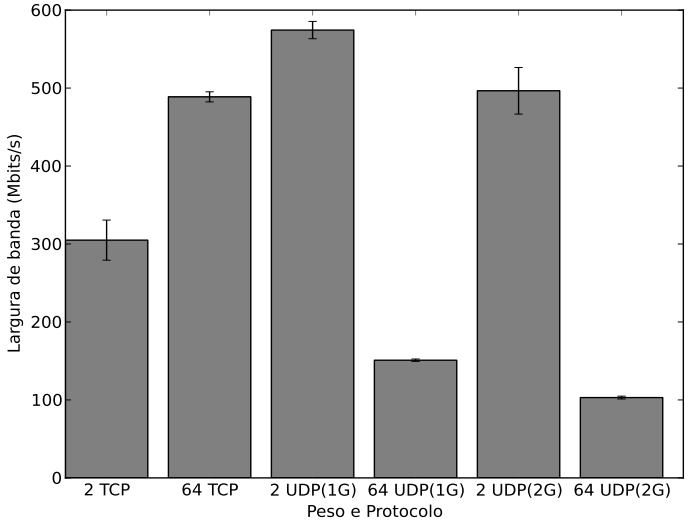

Figura 4.1: *Largura de banda na Recepção de pacotes*

Continuando a análise da Figura 4.1, com protocolo *UDP* e banda de envio de 1 Gbits/s e 2 Gbits/s, percebemos que a largura de banda recebida é maior com peso igual 2, isso acontece pois o sistema entra em *polling* e passa a processar os pacotes que chegam sem interrupções como é possível ver pela quantidade de interrupções geradas na Figura 4.2. Em adição, como o protocolo *UDP* obriga apenas a recepção de pacotes, diferente do TCP que obriga também a responder ao remetente, todo o processo é gasto na recepção de pacotes o que faz o sistema entrar e se manter em *pooling* antes do processador chegar ao limite.

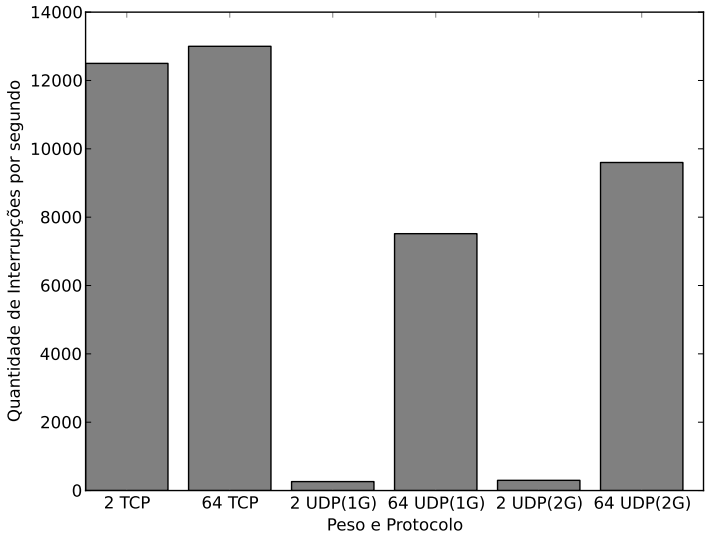

Figura 4.2: *Quantidade de interrupções por segundo no dipositivo de rede virtual durante a recepção de pacotes*

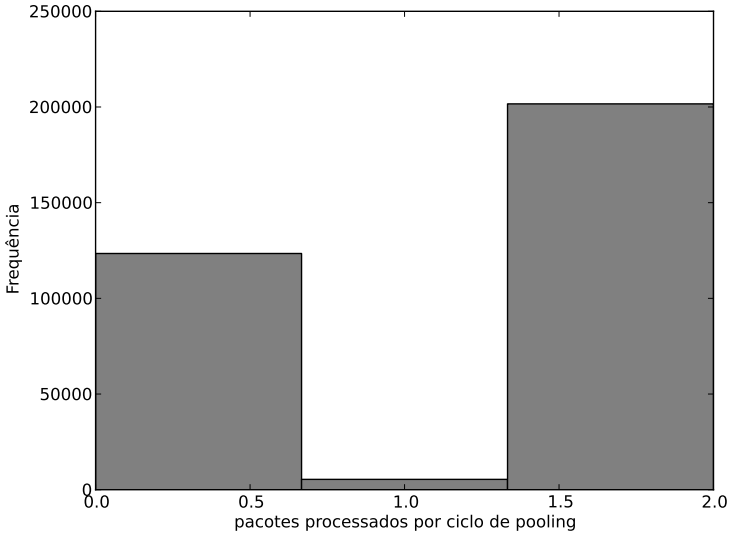

Figura 4.3: *Frequência de pacotes processados no ciclo de polling com peso igual a 2 e protocolo TCP*

Já com peso igual a 64, o sistema não consegue pacotes o suficiente para se manter em *polling* e passa a ter que entrar/sair da fila de *polling* gerando muitas interrupções e reduzindo seu desempenho. Nota-se que nos experimentos com 1GBits/s e 2Gbits/s, o primeiro teve um desempenho um pouco superior em relação ao último, isso é devido ao processador estar no seu limite da sua carga e por causa da chegada de mais pacotes, este passa a ter que rejeitar os que chegam gastando mais processamento.

Na Figura 4.6, vemos que a quantidade de pacotes processada por ciclo de *polling* com peso igual a 64 e protocolo UDP raramente atinge 64, e assim, a recepção nunca entra em *polling*. Por outro lado, na Figura 4.5, onde mostra o mesmo gráfico com peso igual a 2, vemos que o sistema processa, com frequência a quantidade máxima de pacotes, e, como consequência, entra em longos períodos de *pooling*.

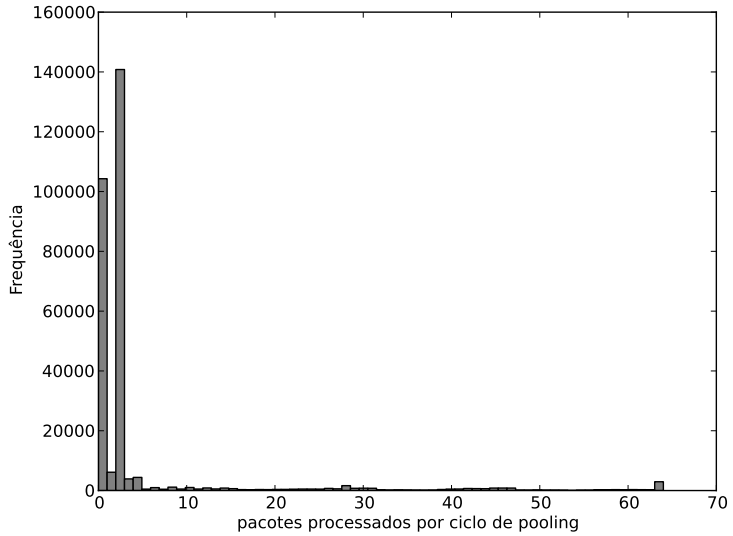

Figura 4.4: *Frequência de pacotes processados no ciclo de polling com peso igual a 64 e protocolo TCP*

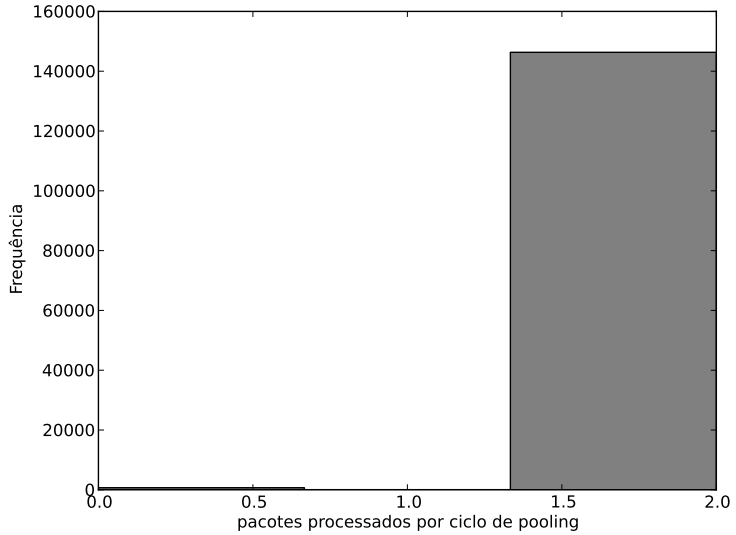

Figura 4.5: *Frequência de pacotes processados no ciclo de polling com peso igual a 2 e protocolo UDP com banda de transferência de 1Gbit/s*

Na Figura 4.7, é mostrada a largura de banda transferida. É notado que a variação no peso não afeta a banda transferida, pois a *NAPI* é usada apenas na recepção de pacotes. Nota-se que apesar do *TCP* ser mais complexo que o*UDP*, a forma como é feita a transmissão de pacotes no *iperf* faz o sistema com *UDP* gerar mais interrupções que o com *TCP* como é possível ver na Figura 4.8, consequentemente, o sistema com *TCP* tem um desempenho melhor na transmissão.

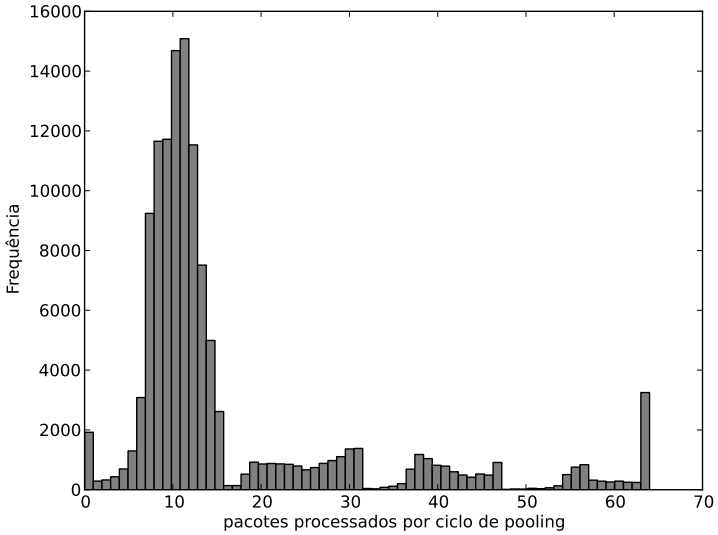

Figura 4.6: *Frequência de pacotes processados no ciclo de polling com peso igual a 64 e protocolo UDP com banda de transferência de 1Gbit/s*

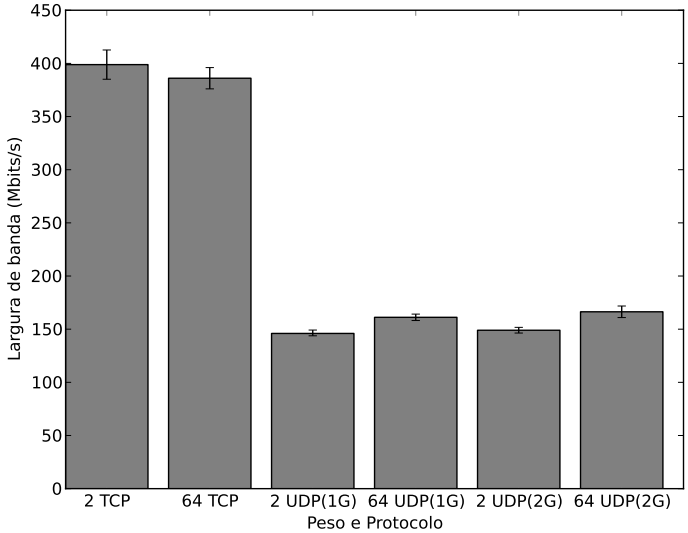

Figura 4.7: *Largura de banda na Transmissão de pacotes*

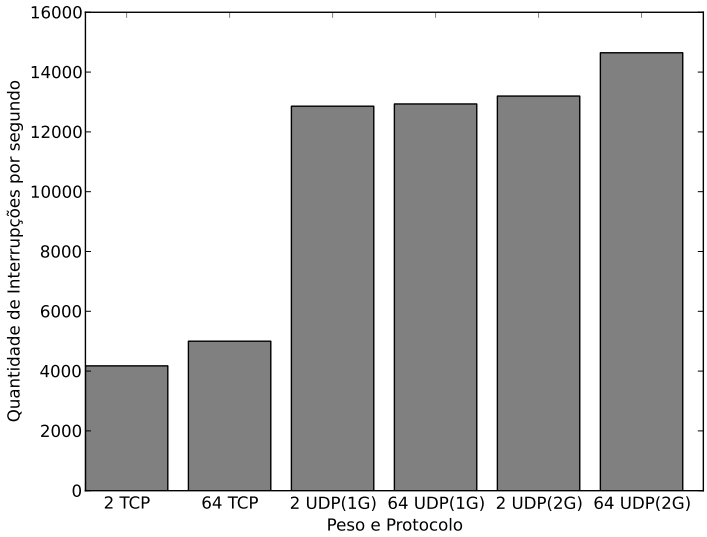

Figura 4.8: *Quantidade de interrupções por segundo no dispositivo de rede virtual durante a transmissão*

Capítulo 5

Proposta

5.1 Tema de Pesquisa

Essa pesquisa tem como tema técnicas de otimização ~~na~~ da virtualização de rede em infraestruturas em nuvem.

5.2 Problema de Pesquisa

O problema a ser tratado nessa pesquisa é o desempenho baixo nas infraestruturas em nuvem que utilizam técnicas de virtualização com *hypervisores* em relação a infraestruturas que virtualizam sem o uso de *hypervisores*, como a virtualização em nível de sistema operacional, ou não virtualizam quando executam aplicações que usam intensamente a rede.

5.3 Evidências do Problema

Para ter evidências que o problema existe, foi feita uma revisão bibliográfica. Nela diversos autores falam sobre problemas na arquitetura da virtualização de rede que é usada pelos *hypervisores* [EF10] [Liu10] [WR12] [Rix08] [STJP08] [ON09] [xen12]. Experimentos foram realizados com infraestruturas usando *XEN* e infraestruturas sem virtualização. O resultado foi que as infraestruturas usando *XEN* tiveram um desempenho muito inferior causado pela alta latência e muito uso do processador [EF10].

5.4 Relevância do Problema

~~Muitas organizações têm investido em computação em nuvem, mais de 150 empresas tem entrado na indústria como fornecedoras de nuvem [Gee09]. No lado dos consumidores de nuvem, uma recente pesquisa com mais de 600 companhias pelo InformationWeek revelou que o número de companhias usando computação em nuvem aumentou de 18% em fevereiro de 2009 para 30% em outubro de 2010 [GED+11].~~ Muitas organizações têm investido em computação em nuvem, até 2009, mais de 150 empresas tornaram-se fornecedoras na indústria de nuvem [Gee09]. No lado dos consumidores de nuvem, uma pesquisa com mais de 600 companhias pelo InformationWeek revelou que o número de companhias usando computação em nuvem aumentou de 18% em fevereiro de 2009 para 30% em outubro de 2010 [GED+11].

Na revisão bibliográfica foram encontrados vários autores que propuseram técnicas para tentar resolver o problema da má utilização dos recursos compartilhados quando há aplicações que fazem uso intenso de rede nas nuvens. [Rix08] [STJP08] [ON09] [LGBK08] [AMN06] [JSJK11] [FA12] [DXZL11].

Uma especificação para dispositivos PCIe chamada *SR-IOV* foi criada apenas para tentar resolver esse problema [Low09] e fabricantes de placas de rede como a Intel[int12] e a Cisco[cis] passaram

a fabricar e vender placas de rede com essa especificação para servidores que usam *XEN* ou *KVM*.

5.5 Proposta de Pesquisa

Na revisão bibliográfica apresentada no Capítulo 3 foram descobertos vários artigos que modificam diferentes partes de uma infraestrutura de nuvem para conseguir um ganho de desempenho. ~~Resolvemos direcionar essa proposta para as estratégias de agregação de interrupções por parecer uma estratégia ainda pouco pesquisada e que pode trazer uma redução grande de interrupções as quais, consequentemente, poderia melhorar o desempenho da infraestrutura de nuvem.~~ Resolvemos direcionar essa proposta para as estratégias de agregação de interrupções por parecer uma estratégia ainda pouco pesquisada e que pode melhorar o desempenho da infraestrutura de nuvem reduzindo a quantidade de interrupções por pacote.

Apesar de não garantir que essa estratégia irá ser melhor que as outras, podemos unir diferentes estratégias para tentar conseguir um ganho ainda maior já que cada estratégia pode modificar uma diferente parte da mesma infraestrutura.

Nessa pesquisa, propomos um algoritmo para agregação de interrupções dos *drivers* das placas de rede virtuais e física tentando ajustar os parâmetros de agregação dinamicamente de forma a garantir uma melhor qualidade dos serviços da aplicação na infraestrutura de nuvem. A qualidade a qual estamos focando está em atributos de desempenho, em específico, a latência, a largura de banda e o uso do processador. Estes variam quando modificamos os parâmetros de agregação. O algoritmo será uma solução se, na infraestrutura, reduzir o uso da *CPU* por pacote na transmissão e recepção e manter a latência consideravelmente baixa.

A estratégia de agregação pode ser feita tanto no *driver* físico como no *driver* virtual de *frontend* e *backend* como foi apresentado na Seção 2.4. Se pensarmos em agregar as interrupções na recepção dos três *drivers* ao mesmo tempo, o comportamento final da rede pode ser um pouco mais complexo de se prever em relação a agregar apenas um dos *drivers*. Isso porque cada *driver* depende tanto dos próprios parâmetros de previsão de chegada de pacotes como também depende dos parâmetros dos *drivers* em que o tráfego de pacotes já passou. O *driver* virtual de *frontend* recebe pacotes que passaram pelo *driver* virtual de *backend*. Sabendo que o *driver* de *backend* espera X pacotes para gerar uma interrupção, esperar receber mais que X pacotes no *frontend* poderia fazer o *driver* ficar esperando demais por pacotes.

Na agregação de interrupções na transmissão, não foram encontrados experimentos, mas em [CRKH05], é mostrado que a quantidade de interrupções devido a transmissão de pacotes é equivalente às interrupções de recepção na virtualização de rede do *XEN*. Também não foi encontrado ainda nenhum artigo que analisa a agregação desses três *drivers* ao mesmo tempo. ~~Uma análise de como os parâmetros estão relacionados e como eles influenciam no tráfego de rede será necessário antes de elaborar um algoritmo.~~

5.5.1 Algoritmo

Na primeira proposta de algoritmo, separaremos as aplicações que estão dentro da nuvem em classes de serviços. Essa idéia foi baseada no algoritmo de moderação de interrupção proposto em [int07]. De acordo com os dados obtidos nessa classificação, iremos variar os parâmetros de agregação de interrupções. A classe de serviços é divida em A, B, C e D. Nas classes A, B e C, a aplicação exige respectivamente uma latência baixa, média, alta. Enquanto que na classe E não exige requisitos de latência.

Com a classificação de todas as aplicações, iremos modificar os parâmetros de agregação em todas as placas de rede. Na placa de rede física, como todo tráfego de todas as máquinas virtuais passam por ela e queremos que os requisitos de todas as aplicações sejam respeitados, modificaremos os parâmetros se baseando na classificação da aplicação que exigir a menor latência. Nas placas de rede virtuais, como elas gerenciam o tráfego apenas da máquina virtual que controla a placa de rede de *backend*, devemos nos preocupar apenas com os requisitos das aplicações que estão dentro dessa

máquina virtual. Assim, modificaremos os parâmetros se baseando na classificação da aplicação que exigir a menor latência e está implantada dentro da máquina virtual que controla a placa de rede de *backend*.

Algoritmo:

1. Analisar os requisitos de latência de todas as aplicações

2. Classificar as aplicações de acordo com esses requisitos

3. Os parâmetros de agregação da placa de rede física serão modificados se baseando na classificação da aplicação que exigir a menor latência. Caso não exista nenhuma aplicação, os parâmetros serão modificados se baseando na classe de serviços E.

4. Os parâmetros de agregação da placa de rede virtual de *backend* e *frontend* serão modificados se baseando na classificação da aplicação que exigir a menor latência dentro da máquina virtual que controla a placa de rede de *backend*. Caso não exista aplicações dentro da máquina virtual, os parâmetros serão modificados se baseando na classe de serviços E.

5.6 Questão de Pesquisa

O algoritmo de agregação de interrupção proposto reduz o uso da *CPU* por pacote na transmissão e recepção e mantêm a latência consideravelmente baixa em relação a infraestrutura sem o algoritmo?

5.7 Cronograma

Revisão bibliográfica adicional: Agosto

Experimento analisando a relação entre cada parâmetro da agregação: Setembro/Outubro

Desenvolvimento de um protótipo de simulador para avaliar as propostas: Setembro/Outubro/-Novembro

Experimento avaliando o algoritmo proposto: Novembro/Dezembro

Escrita de artigo para o SBRC 2012: Novembro/Dezembro

Texto da dissertação: Janeiro/Fevereiro

Escrita de artigo para o CAMAD ou para o WPerformance (simulador): Fevereiro

Capítulo 6

Experimentos

Para analisar o desempenho da *NAPI* em dispositivos de rede virtuais, foram feitos experimentos variando o parâmetro limite do *driver* de rede e1000 da *Intel* em vários *hypervisors* diferentes. Esse *driver* implementa *NAPI* e tem o código-fonte claro e bem escrito, sendo usado por vários *hypervisors* como *XEN*, *VirtualBox*, *VMWare* e *KVM* para processar a recepção e transmissão de dados pela rede na máquina virtual. Outras soluções que os *hypervisors* teriam são os *drivers* de paravirtualização do *XEN* e do Virtio. Ambos têm um desempenho superior ao e1000 por usarem paravirtualização, porém, são mais complexos porque quebram a transparência com a máquina física.

Na sua implementação de recepção de pacotes, o limite define a quantidade limite de pacotes que a tarefa de recepção poderá coletar. Se a quantidade de pacotes atingir esse limite ou esgotar o tempo limite de espera de pacotes, a tarefa é recolocada na fila de *polling*, caso contrário, ela é removida da fila.

6.1 Banda X Limite

Nesse primeiro experimento foi analisado o comportamento da banda em relação a variação do protocolo e a variação do limite. Como *hypervisor* foi usado o *VirtualBox* por sua instalação ser rápida e sua interface ser simples e clara. A máquina física contém um processador i7-2620M de dois núcleos e quatro fluxos de execução, 8 *Gigabytes* de memória *RAM* e sistema operacional *Mac OS X* 10.6.8 enquanto que a máquina virtual usa dois fluxos de execução, 5 *Gigabytes* de memória *RAM* e sistema operacional *Ubuntu* 11.10 com núcleo *Linux* 3.0.43.

Foram analisados a largura de banda na recepção do `iperf` e a quantidade total de pacotes processada pelo *driver*. O limite variou de 1 a 10 de um em um, de 10 a 100 de dez em dez e com o valor 200. A banda de transmissão foi a máxima que a máquina poderia transmitir, no *TCP* dependeria do seu mecanismo de controle de fluxo e no UDP foi 800MBits/s. A escolha do intervalo reduzido quanto menor o limite foi feita considerando que as interrupções físicas irão reduzir significantemente nos primeiros limites e muito pouco nos próximos, sendo assim, é esperado que variações na banda ocorram com valores de limite baixos. Também foram considerados dois artigo [cor05] e [SEB05] que sugerem um melhor desempenho para valores baixos de limite. O maior limite sendo 200 foi definido considerando que o valor inicial é 1 e o valor padrão do *driver* ser 64, sendo a diferença entre eles igual a 63 e como não é esperada encontrar muita variação entre 100 e 200, o valor escolhido foi 200. A banda foi medida usando o programa `iperf` com protocolo *TCP* e *UDP* durante 30 segundos e a quantidade de pacotes pelo `ifconfig`.

A Figura 6.1 mostra a largura de banda da recepção do `iperf` usando *TCP*. Percebe-se que conforme aumenta o limite de 1 até 8, maior a largura de banda. Já entre 9 e 200, a banda varia pouco. Na Figura 6.2, é mostrada a quantidade de pacotes recebida pelo *driver*. Nota-se que existe uma semelhança grande entre a Figura 6.2 e 6.1. Uma correlação linear foi feita com as informações a largura de banda e a quantidade de pacotes recebida. O resultado foi de 0.9999, mostrando que a banda e a recepção de pacotes possuem uma forte correlação.

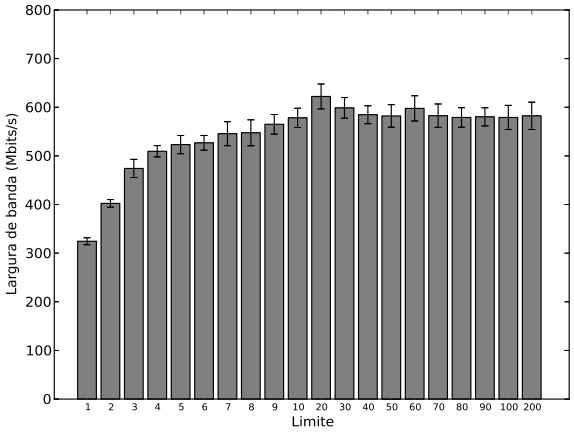

Figura 6.1: *Largura de banda na Recepção de pacotes com protocolo TCP*

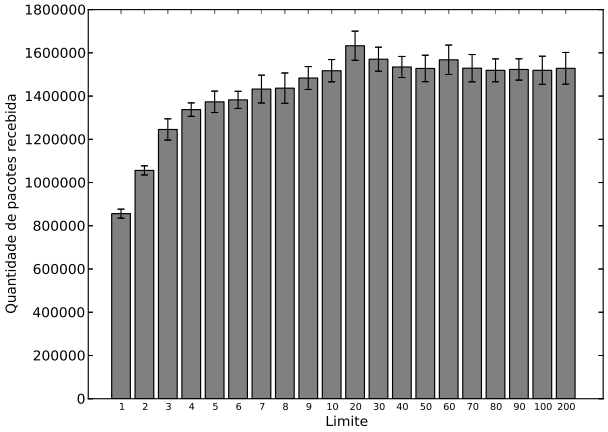

Figura 6.2: *Quantidade de pacotes recebida pelo driver com protocolo TCP*

A Figura 6.3 mostra a banda da recepção usando UDP. Percebe-se um comportamento muito diferente em relação ao *TCP* visto na Figura 6.1, enquanto no *TCP*, o aumento no limite elevaria a banda, o contrário parece ocorrer no UDP, quanto menor o limite maior a banda. Nota-se também um resultado incomum, a banda do UDP apresenta resultados muito inferiores em relação ao *TCP*. Em teoria, o *TCP* tem mecanismos para controle de tráfego e entregas confiáveis de pacotes que deixam o processo de pacotes mais lento em relação ao UDP.

Na Figura 6.4, é mostrada a quantidade de pacotes recebida usando *UDP*, nota-se que a recepção se diferencia completamente em relação largura de banda da recepção. A correlação linear entre elas foi de -0.9999, ou seja, quanto mais pacotes são processados no dispositivo, menor a largura de banda da recepção totalmente contrária ao resultado usando *TCP*. Existe uma provável resposta para essa diferença: os pacotes são processados em grandes quantidades no *driver* com exceção do limite igual a 1 que processa poucos pacotes. Entre a chegada no `iperf` e o processamento do pacote pelo *driver* ocorreu um gargalo devido ao excesso de chegada de pacotes que fez o `iperf` perder muitos pacotes, com exceção do experimento com limite igual a 1 que recebeu poucos pacotes nesse intervalo.

Sendo mais específico, esse gargalo ocorreu no *buffer* de recepção do `socket` criado pelo *iperf*. Aumentando esse *buffer*, percebermos um aumento expressivo da banda como é visto nas Figuras 6.5 e 6.6. A correlação linear foi de 0.9426 uma correlação um pouco abaixo do *TCP*, porém ainda

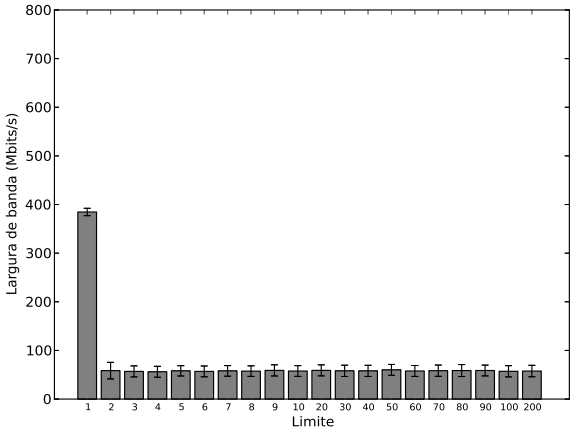

Figura 6.3: *Largura de banda na Recepção de pacotes com protocolo UDP*

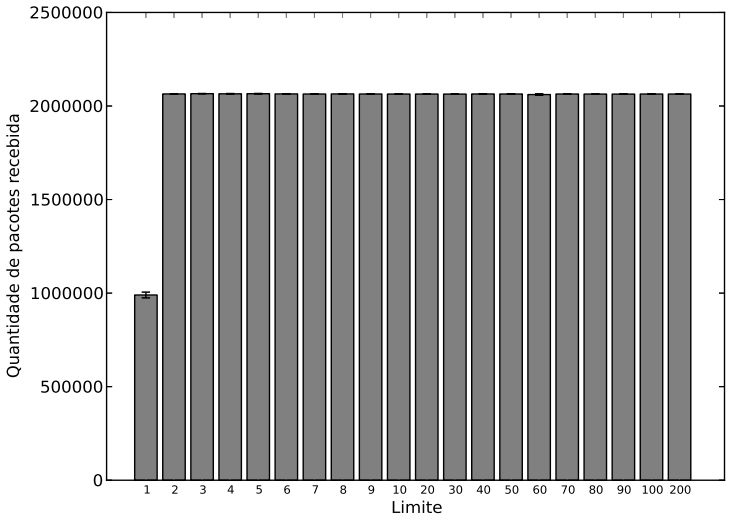

Figura 6.4: *Quantidade de pacotes recebida pelo driver com protocolo UDP*

muito forte. Talvez seja possível achar uma configuração de buffer que se correlacione melhor, mas não é o foco da pesquisa. Assim, tanto com *TCP* como com UDP se pode obter uma largura de banda alta com valor de limite alto e ajustes no *buffer*.

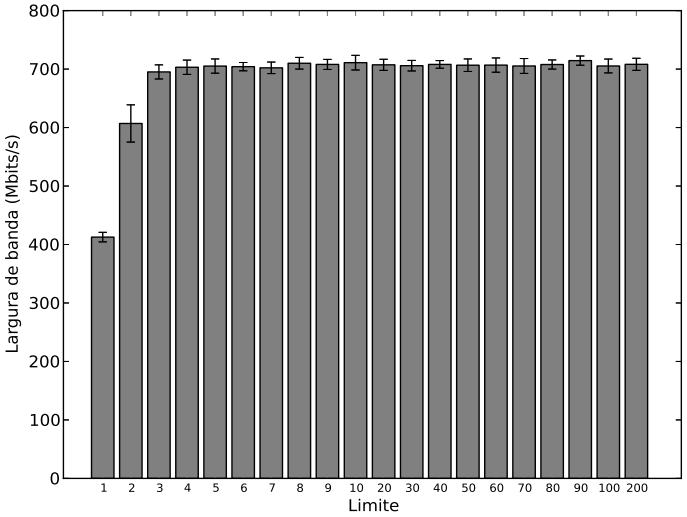

Figura 6.5: *Largura de banda na Recepção de pacotes com protocolo UDP modificando o buffer de recepção*

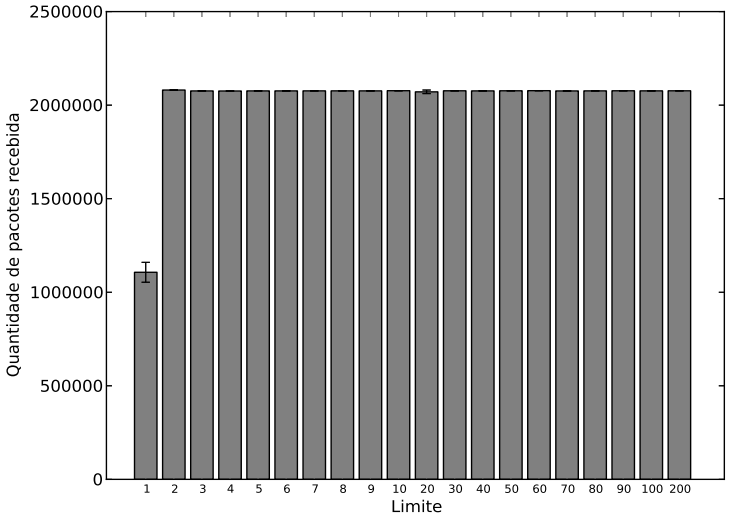

Figura 6.6: *Quantidade de pacotes recebida pelo driver com protocolo UDP modificando o buffer de recepção*

6.2 Interrupções

Nesse segundo experimento foram realizados experimentos com o limite em relação as interrupções de *software* e *hardware* com o protocolo UDP. Como *hypervisor* foram usados o *VirtualBox*, o *VMware* e o *Xen*.

No experimento com o *VirtualBox* e *VMware* foram usadas as mesmas configurações que o experimento anterior. No *XEN*, a máquina física contém um processador i7 Ivy bridge de quatro núcleos e oito fluxos de execução, 16 *Gigabytes* de memória *RAM* e sistema operacional *Ubuntu* 12.10 enquanto que a máquina virtual usa dois fluxos de execução, 5 *Gigabytes* de memória *RAM* e sistema operacional *Ubuntu* 11.10 com núcleo *Linux* 3.0.12.

Foram variados a largura de banda de transmissão de 100 até 1000 Mbits/s de cem em cem e o limite em 1, 2, 60 e 200. O Tamanho do *buffer* de recepção foi definido para 8 Mbytes. Foram medidos a largura de banda de recepção do `iperf`, a quantidade de pacotes recebida usando o `ifconfig`, a quantidade de interrupções de hardware, o uso de *CPU* total, e o uso de *CPU* pelas interrupções de software usando `sar`.

6.2.1 VirtualBox

Nos experimentos com o *VirtualBox*, as Figuras 6.7, 6.8, 6.10, 6.9 e 6.11 mostram respectivamente quantidade de interrupções gerada pelo dispositivo de rede, o uso de *CPU* pelas interrupções de software, a quantidade de pacotes recebida pelo *driver*, o uso da *CPU* e a banda de recepção.

Na Figura 6.7, em todas as curvas vemos que a quantidade de interrupções tem um intervalo de crescimento seguido de uma queda. Quanto menor o limite, mais rápida é a queda e com menos banda ela começa a decrescer. Com o limite igual a 1 a queda começa com banda de 200 Mbits/s e em 500 Mbits/s o núcleo do sistema entra em *pooling* ignorando todas as interrupções. O mesmo ocorre com limite igual a 2 iniciando a queda um pouco mais fraca com banda igual a 200 Mbits/s e nos limites de 60 e 200 com banda igual a 700 Mbits/s, mas o núcleo do sistema não entra em *pooling* porque a banda de transmissão máxima é de 800 Mbits/s.

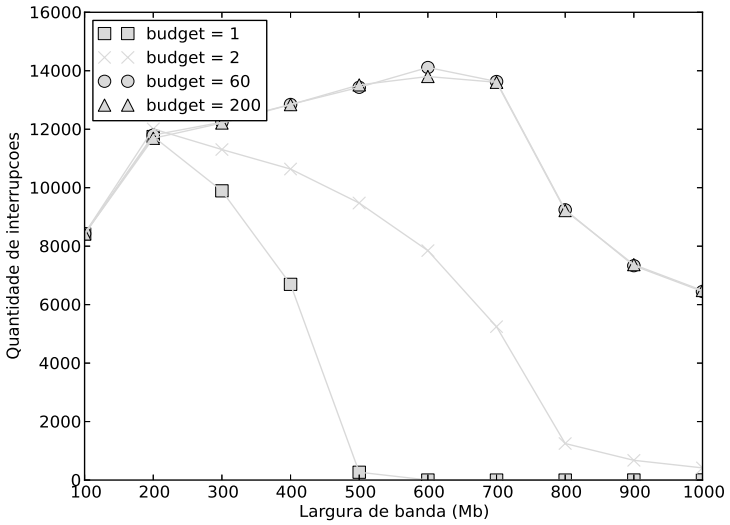

Figura 6.7: *quantidade de interrupções de hardware gerada pela placa de rede virtual no VirtualBox*

Analisando a Figura 6.8, nas curvas com limites iguais a 60 e 200, nota-se que o gráfico cresce e atinge seu máximo (aproximadamente 40%) em 300, se mantêm constante de 300 até 700 Mbits/s, decresce rapidamente de 700 até 800 e levemente decresce de 800 até 1000. Com limite igual a 1, verifica-se um grande uso de *CPU* em relação aos demais, chegando a 50%, quando a banda é maior que 500 Mbits/s, coincidindo com o momento que as interrupção de *hardware* chegaram a

0. Nota-se também que diferente das outras curvas que tem um decremento rápido de 700 até 800 Mbits/s, essa curva não decrementa, provavelmente porque a quantidade de interrupções se tornou mínima em 500 MBits/s. Por fim, a curva igual a 2 se apresenta como uma intermediária entre as curvas de limite igual 1 e 60/200, possui um comportamento semelhante as curvas 60/200 com o núcleo usando um pouco mais de *CPU*.

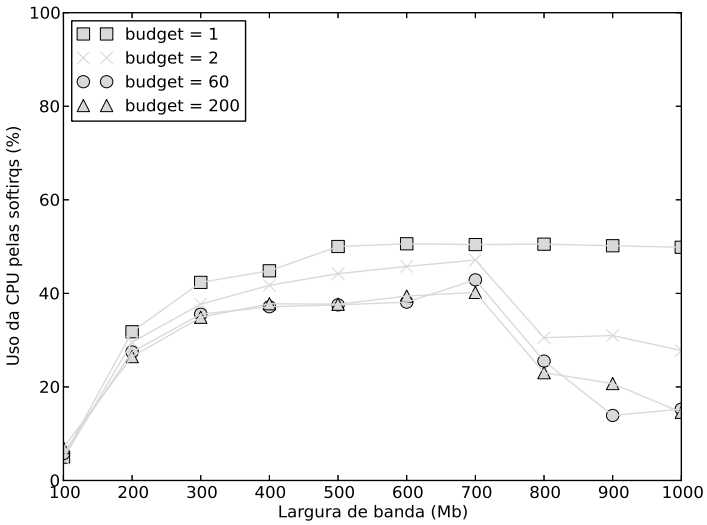

Figura 6.8: *uso da CPU pelas interrupções de software no VirtualBox*

A Figura 6.9 está relacionada com o comportamento da Figura 6.8 e com o uso da *CPU* pelo `iperf`. Nota-se que a *CPU* parece não chegam ao gargalo de 2 fluxos de execução, porém, não foi considerada a sobrecarga devido a emulação que pode ser medida através da máquina física.

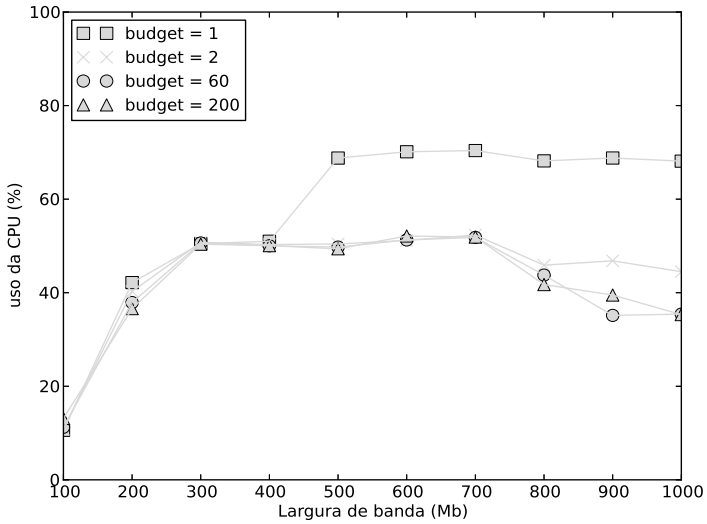

Figura 6.9: *uso da CPU no VirtualBox*

Na Figura 6.10, percebe-se que todas as curvas começam crescente e num determinado momento, elas ficam constantes limitada por algum fator. A curva com limite igual a 1 passa a ficar constante quando a banda é maior que 500 MBits/s, exatamente onde as interrupções chegam próximas de 0 e as interrupções de software chegam ao máximo. Já as outras curvas passam a ficar constantes

somente em 800 Mbits/s que é o limite da banda de transmissão, nesse caso, o *driver* conseguiu processar todos os pacotes.

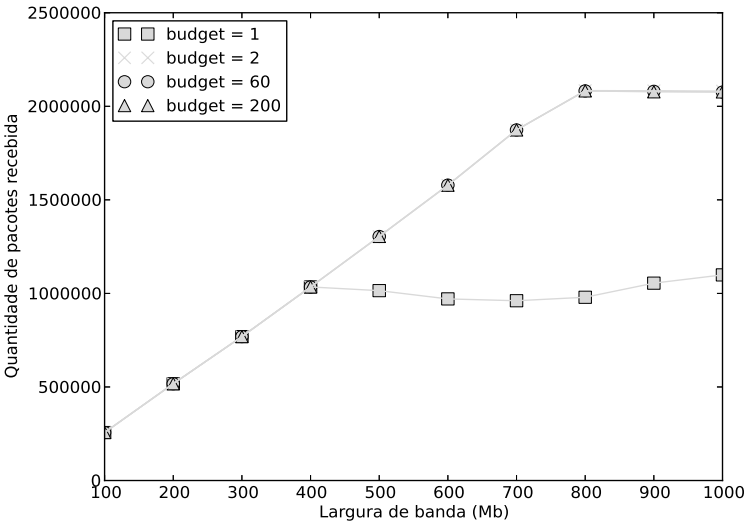

Figura 6.10: *Quantidade de pacotes recebida pelo driver no VirtualBox*

Na última Figura 6.11, a curva com limite igual a 1,

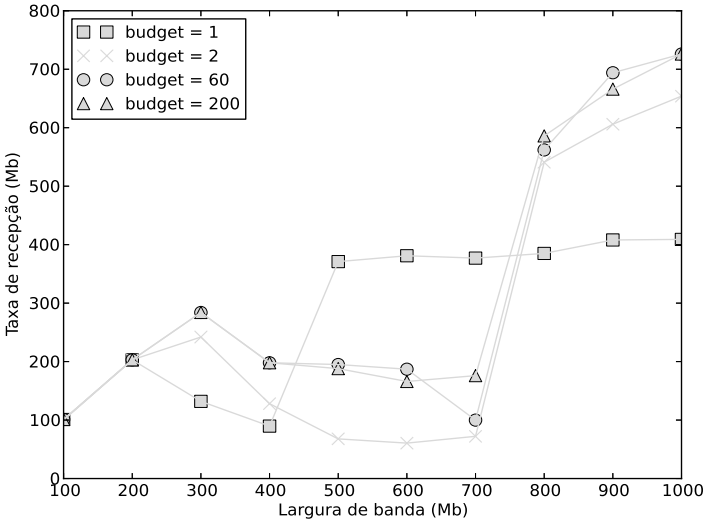

Figura 6.11: *Largura de banda de recepção no VirtualBox*

6.2.2 VMware

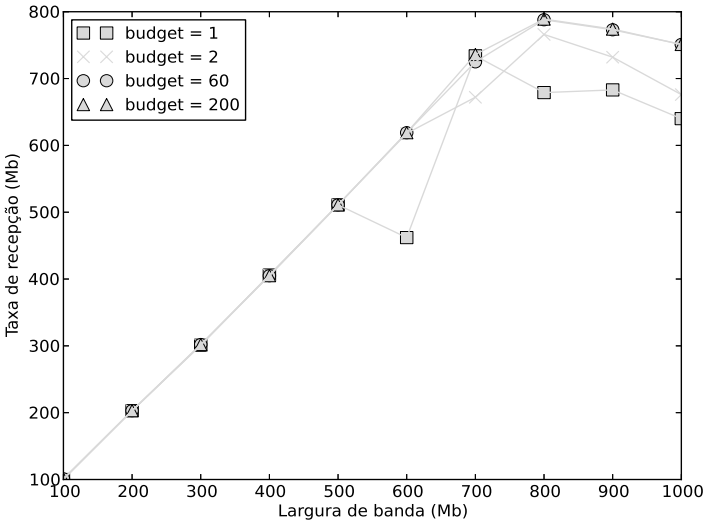

Figura 6.12: *Largura de banda de recepção no VMware*

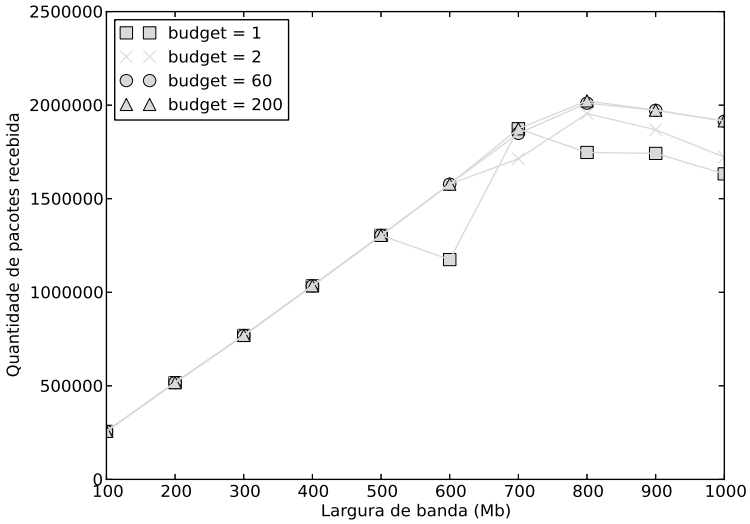

Figura 6.13: *Quantidade de pacotes recebida pelo driver no VMware*

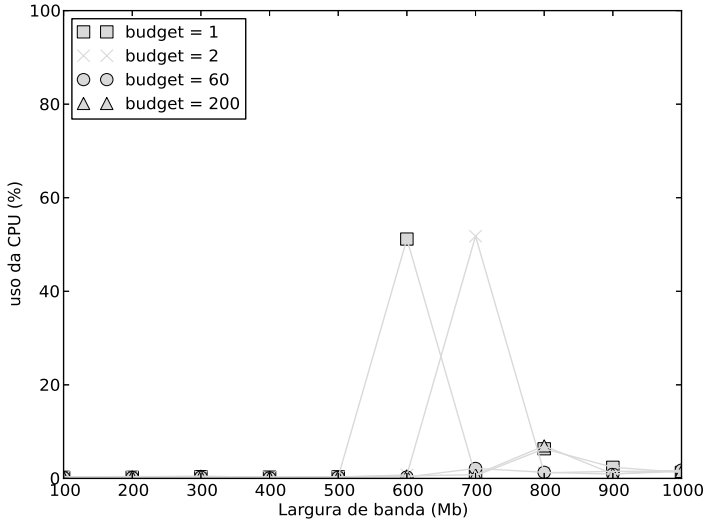

Figura 6.14: *uso da CPU no VMware*

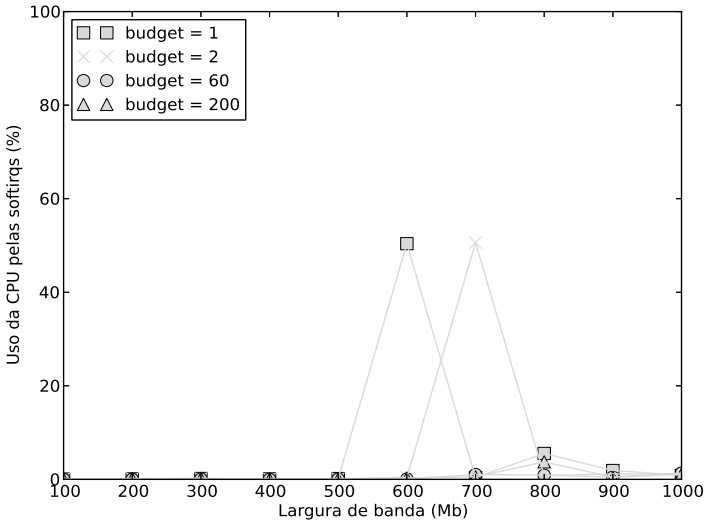

Figura 6.15: *uso da CPU pelas interrupções de software no VMware*

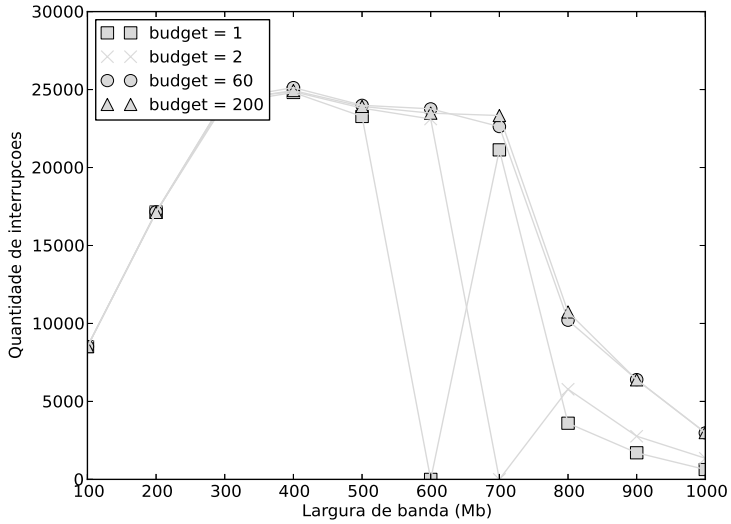

Figura 6.16: *quantidade de interrupções de hardware gerada pela placa de rede virtual no VMware*

6.2.3 Xen

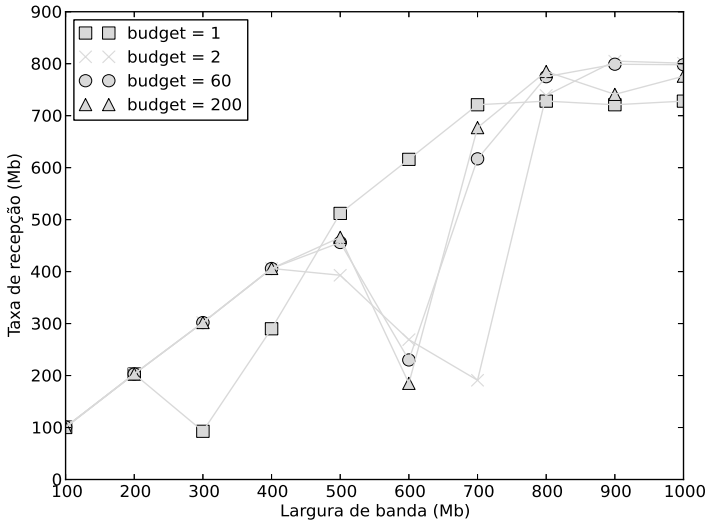

Figura 6.17: *Largura de banda de recepção no Xen*

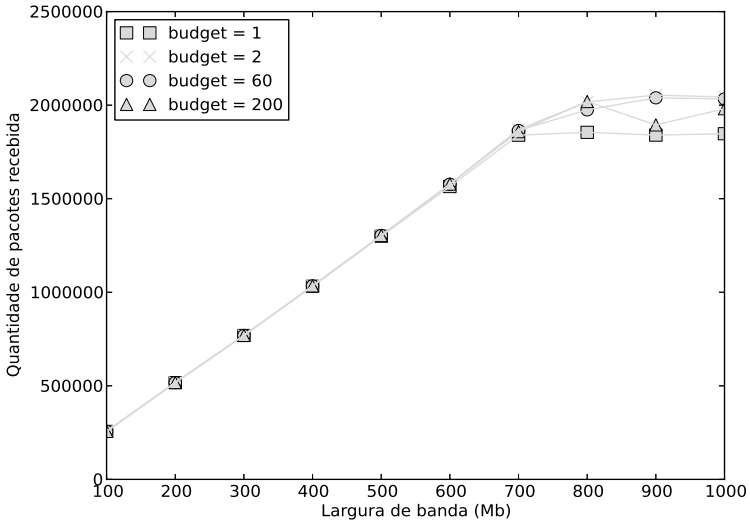

Figura 6.18: *Quantidade de pacotes recebida pelo driver no Xen*

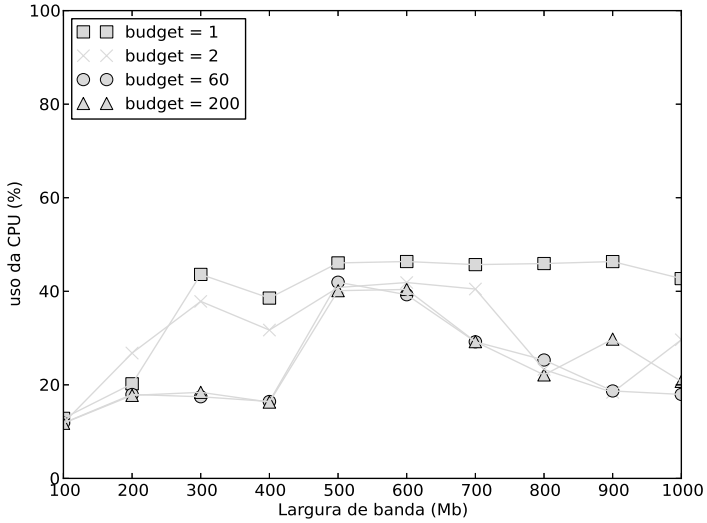

Figura 6.19: *uso da CPU no Xen*

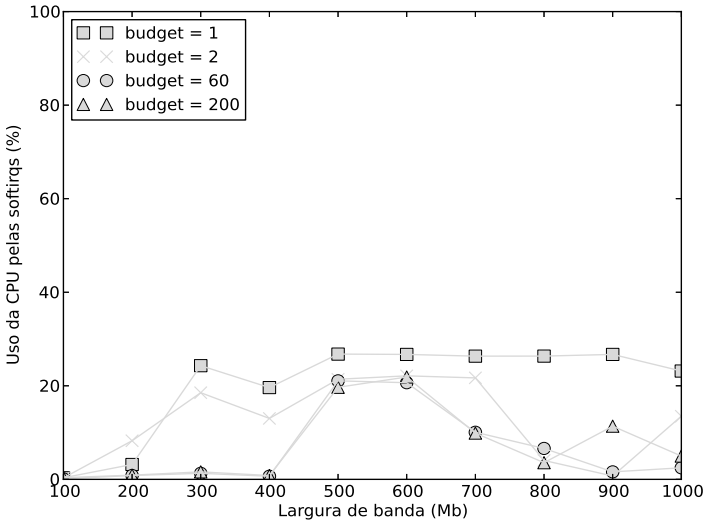

Figura 6.20: *uso da CPU pelas interrupções de software no Xen*

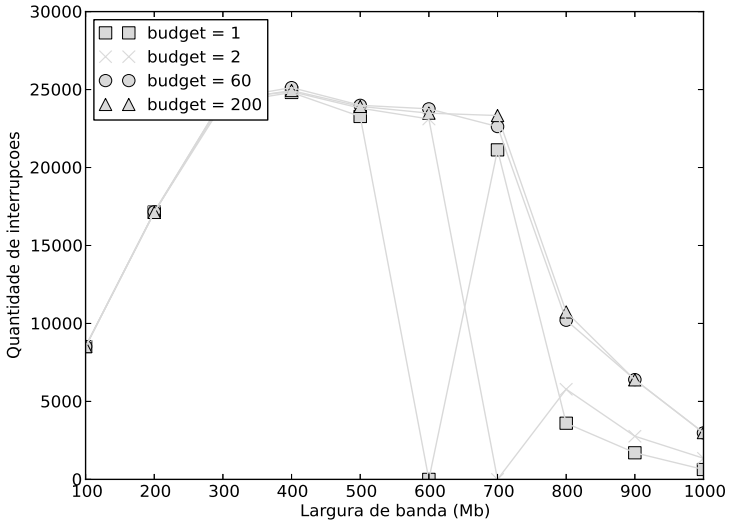

Figura 6.21: *quantidade de interrupções de hardware gerada pela placa de rede virtual no Xen*

Referências Bibliográficas

[AFG+09] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica et al. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009. 1, 3, 5, 6

[AMN06] P. Apparao, S. Makineni e D. Newell. Characterization of network processing overheads in xen. Em *Proceedings of the 2nd international Workshop on Virtualization Technology in Distributed Computing*, página 2. IEEE Computer Society, 2006. 9, 15, 17, 27

[AU09] G.P. Alkmim e J.Q. Uchôa. Uma solução de baixo custo para a migração de máquinas virtuais. Em *VIII WPerformance-Workshop em Desempenho de Sistemas Computacionais e de Comunicaçao-XXIX CSBC-Congresso da Sociedade Brasileira de Computaçao*, 2009. 7

[BDF+03] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt e A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003. 1, 6, 16

[BM99] S. BRADNER e J. MCQUAID. Rfc 2544. *Benchmarking methodology for network interconnect devices*, 1999. 9

[CCW+08] V. Chaudhary, M. Cha, JP Walters, S. Guercio e S. Gallo. A comparison of virtualization technologies for hpc. Em *Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on*, páginas 861–868. IEEE, 2008. 1, 5, 6, 7, 9, 15, 16

[cis] Cisco ucs virtual interface card 1240. http://www.bailey.ciscosolution.net/sw/swchannel/productcatalogcf_v2/internet/model.asp?ProductMasterId=2426700&ParentId=607272. Acessado em: 19/7/2012. 27

[cor05] corbet. Napi performance - a weighty matter. http://lwn.net/Articles/139884/, 2005. Acessado em: 16/7/2012. 11, 21, 31

[Cor09] Jonathan Corbet. Generic receive offload. http://lwn.net/Articles/358910/, 2009. Acessado em: 25/6/2012. 9, 12, 15

[CRKH05] Jonathan Corbet, Alessandro Rubini e Greg Kroah-Hartman. *Linux Device Drivers, 3rd Edition*. O'Reilly Media, Inc., 2005. 11, 21, 28

[CYSL10] J. Che, Y. Yu, C. Shi e W. Lin. A synthetical performance evaluation of openvz, xen and kvm. Em *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific*, páginas 587–594. IEEE, 2010. 6

[DXZL11] Y. Dong, D. Xu, Y. Zhang e G. Liao. Optimizing network i/o virtualization with efficient interrupt coalescing and virtual receive side scaling. Em *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, páginas 26–34. IEEE, 2011. 1, 9, 11, 12, 15, 18, 27

[Eas07] Thomas M. Eastep. Xen network environment. http://www1.shorewall.net/XenMyWay.html, 2007. Acessado em: 25/6/2012. xi, 9

[EF10] J. Ekanayake e G. Fox. High performance parallel computing with clouds and cloud technologies. *Cloud Computing*, páginas 20–38, 2010. 1, 7, 9, 15, 16, 27

[FA12] T. Fortuna e B. Adamczyk. Improving packet reception and forwarding within virtualized xen environments. *Computer Networks*, páginas 153–160, 2012. 9, 15, 18, 27

[GED+11] G.E. Gonçalves, P.T. Endo, T. Damasceno, A.V.A.P. Cordeiro, D. Sadok, J. Kelner, B. Melander e J.E. Mångs. Resource allocation in clouds: Concepts, tools and research challenges. *Simpósio Brasileiro de Rede de Computadores*, 2011. 1, 7, 27

[Gee09] Jeremy Geelan. The top 150 players in cloud computing. http://cloudcomputing.sys-con.com/node/770174, 2009. Acessado em: 16/7/2012. 27

[int07] Interrupt moderation, intel gbe controllers. http://www.intel.com/content/www/us/en/ethernet-controllers/gbe-controllers-interrupt-moderation-appl-note.html, 2007. Acessado em: 29/7/2012. 15, 19, 20, 28

[int12] Intel server adapters. http://www.intel.com/support/network/adapter/pro100/sb/CS-031492.htm, 2012. Acessado em: 19/7/2012. 27

[Jam04] T.Y. James. Performance evaluation of linux bridge. Em *Telecommunications System Management Conference*, 2004. 9

[JSJK11] J.W. Jang, E. Seo, H. Jo e J.S. Kim. A low-overhead networking mechanism for virtualized high-performance computing systems. *The Journal of Supercomputing*, páginas 1–26, 2011. 9, 15, 17, 27

[LGBK08] Guangdeng Liao, Danhua Guo, Laxmi Bhuyan e Steve R King. Software techniques to improve virtualized i/o performance on multi-core systems. Em *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, páginas 161–170, New York, NY, USA, 2008. ACM. 9, 15, 17, 27

[Liu10] J. Liu. Evaluating standard-based self-virtualizing devices: A performance study on 10 gbe nics with sr-iov support. Em *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, páginas 1–12. IEEE, 2010. 1, 7, 15, 17, 27

[Low09] Scott Low. What is sr-iov? http://blog.scottlowe.org/2009/12/02/what-is-sr-iov/, 2009. Acessado em: 19/7/2012. 27

[MCZ06] A. Menon, A.L. Cox e W. Zwaenepoel. Optimizing network virtualization in xen. Em *Proceedings of the annual conference on USENIX'06 Annual Technical Conference*, páginas 2–2. USENIX Association, 2006. 11

[ON09] H. Oi e F. Nakajima. Performance analysis of large receive offload in a xen virtualized system. Em *Computer Engineering and Technology, 2009. ICCET'09. International Conference on*, volume 1, páginas 475–480. IEEE, 2009. 9, 15, 17, 27

[PZW+07] P. Padala, X. Zhu, Z. Wang, S. Singhal, K.G. Shin et al. Performance evaluation of virtualization technologies for server consolidation. *HP Laboratories Technical Report*, 2007. 6

[Rix08] S. Rixner. Network virtualization: Breaking the performance barrier. *Queue*, 6(1):36–ff, 2008. 1, 7, 8, 15, 16, 27

[Sal07] K. Salah. To coalesce or not to coalesce. *AEU-International Journal of Electronics and Communications*, 61(4):215–225, 2007. 1, 11, 15, 19, 20

[SBB+07] Galen M. Shipman, Ron Brightwell, Brian Barrett, Jeffrey M. Squyres e Gil Bloch. Investigations on infiniband: Efficient network buffer utilization at scale. Em *Proceedings, Euro PVM/MPI*, Paris, France, October 2007. 9, 15

[SBdSC] A.H. Schmidt, M.P. Boufleur, R.C.M. dos Santos e A.S. Charao. Análise de desempenho da virtualizaç ao de rede nos sistemas xen e openvz. 6

[SEB05] K. Salah e K. El-Badawi. Analysis and simulation of interrupt overhead impact on os throughput in high-speed networks. *International Journal of Communication Systems*, 18(5):501–526, 2005. xi, 15, 18, 19, 20, 31

[Spe10] Stephen Spector. New to xen guide. http://www.xen.org/files/Marketing/NewtoXenGuide.pdf, 2010. Acessado em: 16/7/2012. 9

[Sta10] W. Stallings. *Computer organization and architecture*. Pearson Education India, 8 edição, 2010. xi, 1, 2, 4, 5

[STJP08] Jose Renato Santos, Yoshio Turner, G. Janakiraman e Ian Pratt. Bridging the gap between software and hardware techniques for i/o virtualization. Em *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ATC'08, páginas 29–42, Berkeley, CA, USA, 2008. USENIX Association. xi, 7, 9, 10, 15, 17, 27

[WR12] C. Waldspurger e M. Rosenblum. I/o virtualization. *Communications of the ACM*, 55(1):66–73, 2012. 1, 7, 9, 15, 16, 27

[xen12] Xen best practices. http://wiki.xen.org/wiki/XenBestPractices, 2012. Acessado em: 25/6/2012. 27