Otimização da Mesclagem de Interrupções em Infraestruturas em Nuvem

Eduardo Hideo Kuroda

Qualificação 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 Ciências da Computação

Orientador: Prof. Dr. Daniel Macêdo Batista

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

São Paulo, julho de 2012

Otimização da Mesclagem de Interrupções em Infraestruturas em Nuvem

Proposta de dissertação apresentada ao
Instituto de Matemática e Estatística da Universidade de São Paulo
como requisito parcial para qualificação de
Mestre em Ciência da Computação.

Resumo

As infraestrutura em nuvem se comportam com um desempenho baixo se comparado com infraestruturas sem virtualização, principalmente quando pensamos em implantar aplicações intensivas de rede.

Uma das principais causas é a arquitetura da virtualização de rede. Diferente da arquitetura de rede padrão, há alguns passos adicionais para transmitir e receber um pacote de informação dentro de uma máquina virtual o que implicam em um custo adicional tanto na memória como no processamento.

Para reduz o custo do processamento, uma ideia é mesclar 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 mesclagem de interrupções.

Nessa pesquisa criaremos um algoritmo para otimizar a mesclagem de interrupções tentando garantir uma melhor qualidade dos serviços dentro da infraestrutura.
Palavras-chave: computação em nuvem, virtualização de rede, mesclagem de interrupções

Abstract

The cloud infrastructure behaves underperforming compared to infrastructure without virtualization, especially when we think of deploying network-intensive applications.

One major cause is the architecture of network virtualization. Unlike the standard network architecture, there are some additional steps to transmit and receive a packet of information within a virtual machine, which imply an additional cost in both memory and the processing.

To reduce the cost of processing, one idea is coalesce multiple interrupts generated when a packet is sent or received. This would reduce the amount of processing a packet, but also increase the latency. This strategy is called coalescing interruption.

This research will create an algorithm to optimize the interruption coalescing trying to ensure a better quality of services within the infrastructure.

Keywords: cloud computing, network virtualization, interrupt coalesing

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*)
IaaS	infraestrutura como um serviço (*Infrastructure as a service*)
IP	protocolo de Internet (*Internet Protocol*)

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Numa infraestrutura de nuvem, os recursos são controlados pelo fornecedor e alocados de maneira elástica de acordo com a necessidade do consumidor [GED+11]. Para obter essa 'elasticidade', normalmente, a nuvem adotada uma tecnologia chamada virtualização.

A virtualização divide um recurso poderoso em recursos menores chamados de máquinas virtuais [BDF+03]. Com recursos menores, é possível fornecer ao consumidor uma quantidade menor de recursos computacionais que ainda satisfaçam seus requisitos e também alocar mais sob demanda [AFG+09].

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* e sua arquitetura de virtualização de rede. Diferente 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 reduz o custo do processamento, uma ideia é mesclar várias interrupções geradas quando um pacote é enviado ou recebido. Isso reduziria a quantidade de processamento por pacotes [AMN06] [CCW+08].

Assim, essa pesquisa foca em automatizar a configuração de mesclagem de interrupções dos *drivers* de rede, esta gerada pelo *hypervisor*, tentando ajustar esse parâmetro dinamicamente de forma a garantir uma melhor qualidade do serviço da infraestrutura.

1.1 Considerações Preliminares

Esse trabalho está direcionado para contribuições em específico na área de computação em nuvem e redes pelo interesse do autor.

1.2 Objetivos

O objetivo dessa pesquisa é demostrar que o algoritmo de mesclagem de interrupções reduz o uso da *CPU* por pacote na transmissão e recepção e mantém a latência consideravelmente baixa.

1.3 Contribuições

As principais contribuições desse trabalho são:

- Fornecer um algoritmo capaz de garantir uma melhor qualidade na infraestrutura de nuvem para diferentes tipos de aplicações.

- Automatizar as configurações de mesclagem de interrupções do *driver* da placa de rede física e virtual de acordo com a transmissão e recepção de pacotes.

1.4 Organização do Trabalho

No Capítulo 2, apresentamos os conceitos de Virtualização de servidores, Virtualização de E/S, Virtualização de rede, Mesclagem de interrupções e Computação em nuvem. No Capítulo 3, apresentamos uma revisão bibliográfica na área de virtualização de rede. Finalmente, no Capítulo 4, apresentamos uma proposta na área de virtualização de rede.

Capítulo 2

Conceitos

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 das CPDs (Centro de Processamento de Dados) que fornecem os serviços [AFG+09].

Como o termo nuvem é muito abrangente, ele foi dividido em várias classificações [AFG+09], entre elas, o tipo de serviço o qual fornecem. Nesse texto, quando falamos de nuvem, estaremos nos referindo a serviços que fornecem uma infraestrutura (*IaaS*). Cada infraestrutura pode receber várias requisições de hospedar programas de desenvolvedores e, nesse caso, terá que implantá-los em algum local no interior dela. 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 e CPDs, que contêm e controlam várias máquinas virtuais (MV) usando alguma técnica de virtualização. Cada requisição para implantar ou executar um programa é feita oferecendo as máquinas virtuais as quais estão dentro de um nó da infraestrutura.

2.1.1 Uso da virtualização

Na computação em nuvem, em particular quando se é fornecido uma infraestrutura para implantar aplicações(*IaaS*), a adoção da virtualização melhora a utilização dos recursos e protege o servidor de problemas que os softwares dos clientes possam causar em relação a servidores com máquinas puras [CCW+08].

Como consequência, também permite 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 a configuração de uma máquina virtual pode ser ampliada 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.

2.2 Virtualização

2.2.1 Virtualização de servidores

Os servidores normalmente são constituídos de CPDs que estão ligados de alguma forma por uma rede. Quando queremos fornecer recursos de maneira eficiente, uma tecnologia popularmente utilizada é a virtualização [GED+11]. A virtualização divide um computador, geralmente com grande capacidade de processamento, em recursos menores chamadas 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 também alocar mais 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 *hypervisores* estão o *KVM*, o *XEN*, o *VMWare* e o *VirtualBox*.

Diferente 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. Nele, o hardware virtual consegue conversar diretamente com o dispositivo emulado. Isso garante uma carga extra 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 operacional 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*.

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*.

A virtualização completa e nativa tem uma grande vantagem em relação as outras: não é necessário alterar o núcleo do sistema. Isto as tornam mais simples e mais portável já que sistemas operacionais com código fechados podem usar elas. A para-virtualização e a virtualização em nível de sistema operacional exigem uma modificação no núcleo, 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].

2.2.2 Virtualização de dispositivos de E/S

Com a virtualização de servidores, os dispositivos de E/S físicos passam a ter que sofrer modificações já que em um servidor não tem apenas um único sistema operacional, mas sim, várias máquinas virtuais com um sistema dentro de cada uma.

[Rix08] separou 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 para várias MV.

Comparando a virtualização de E/S privada com a compartilhada há uma subutilização na virtualização privada, pois parte do tempo em que a MV não usa o dispositivo é desperdiçada. Por outro lado, o desempenho da virtualização compartilhada é pior já que divide o recurso com outras máquinas.

Quando pensamos em escalar 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, queremos 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].

Para contornar esse problema, normalmente, *hypervisores* como *XEN*, *KVM* e *VMWare* restringem o acesso a um dispositivo físico para apenas uma máquina virtual e o acesso a esse dispositivo pelas outras máquinas virtuais é feito através dessa máquina. Essa restrição traz uma perda de desempenho em relação a ambientes que não usam virtualização quando o uso da rede é intensa, por exemplo [CCW+08] [EF10] [Liu10].

[WR12] fez algumas menções sobre o uso de técnicas de virtualização de E/S. Dentre as vantagens, ele cita a melhor utilização dos recursos e a economia de custos em relação a sistemas que estão com a implementação física acoplada com o dispositivo lógico, pois vários sistemas podem aproveitar o mesmo recurso. 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. A funcionalidade de suspender, migrar e resumir uma máquina virtual também é possível, pois com o dispositivo lógico desacoplado da implementação física, 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 requisições que passam pelo dispositivo lógico sejam transformadas. Em um exemplo de leitura e escrita no disco, além de simplesmente ler/escrever no disco, é possível guardar uma cópia da informação antiga como cópia de segurança para conseguir num momento futuro, "viajar no tempo"e desfazer algo que foi feito. Outra ideia é criptografar a informação quando alguém escrever no disco, dificultando outras pessoas de acessarem o seu conteúdo escrito.

2.2.3 Virtualização da Rede

A virtualização de rede que também é um dispositivo de E/S tem algumas particularidades em relação a outros dispositivos. Segundo [Rix08], Comparando a virtualização de E/S com a virtualização de rede, a complexidade de virtualizar a rede é muito maior pelo fato de não se conhecer o destino de uma informação e a 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.

2.2.4 Virtualização da Rede no XEN

Apesar da implementação dada ser específica para o *XEN*, outros *hypervisores* como *WMware* e *KVM* adotam uma arquitetura semelhante [STJP08].

Dando uma breve explicação sobre os termos que o *XEN* usa: O *dom0* ou domínio zero é a primeira máquina virtual iniciada pelo *XEN*. Ela tem certos privilégios que as outras máquinas virtuais não têm como iniciar novas máquinas e acessar o hardware diretamente. O `domU` ou domínio 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, temos dois tipos de configuração: ponte e roteador. Ambos são sistemas que servem para encaminhar 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` conseguiriam enviar e receber pacotes do dispositivo de rede com o `dom0` como intermediário.

[Eas07] descrevem a implementação da configuração de ponte na qual uma ponte virtual(`xenbr0`) é criada dentro do `dom0` como é possível ver na Figura 2.1. Essa ponte está ligada na interface de

rede física pela porta `peth0`. A porta `vif0.0` está sendo usada para tráfegos de/para `dom0` e as portas `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`.

Na configuração de roteador o `dom0` cria uma ligação entre ele e cada `domU`. Rotas de cada `domU` são adicionadas na tabela de roteamento do `dom0`, nesse caso o *IP* de cada *domU* tem que ser estático.

[Jam04] fez 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 pela maioria dos trabalhos relacionados terem feito experimentos com essa configuração.

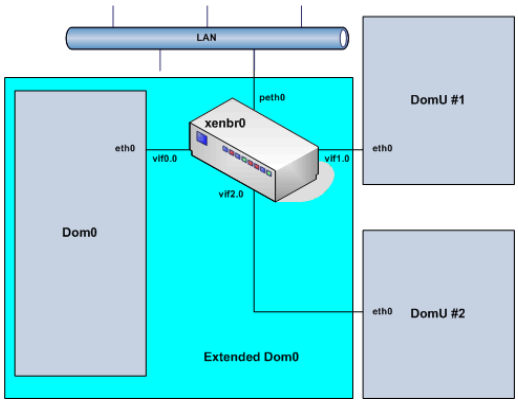

Figura 2.1: *ponte virtual criada no XEN [Eas07]*

Na Figura 2.2 vemos a arquitetura da virtualização da rede usando ponte no *XEN* segundo [STJP08].

Para transmitir/receber um pacote no domínio hospedeiro(`domU`) é usado o canal de E/S. 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 coloca algum pacote nessas 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 alguém de um domínio escreveu no *buffer* de E/S. Isso é feito através de uma interrupção virtual no outro domínio.

O *buffer* de descrição em anel guarda os detalhes de requisições entre o *driver* de *frontend*(*netfront*) que fica no domínio que controla os *drivers*(domínio do *driver*) 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.

Para o domínio do *driver* ter acesso as 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, o `domU` coloca o pacote no *buffer* de E/S, as suas páginas de referência no *buffer* de descrição e notifica o domínio do *driver*. Este 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 a ponte dentro do domínio do *driver*. A ponte então avisa o *netback*

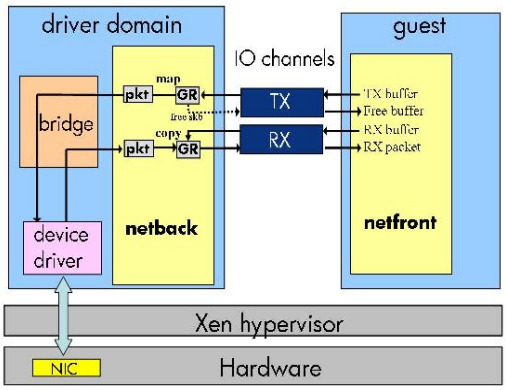

Figura 2.2: *arquitetura da rede virtual no XEN [STJP08]*

correto sobre a chegada de pacotes. O `netback` o copia para uma página da memória que foi fornecida pelo *netfront* e envia uma notificação para o mesmo. Quando o `netfront` recebe a notificação, ele pega oque está no *buffer*, manda para o seu sistema e libera as páginas fornecidas.

2.3 Mesclagem de Interrupções na Recepção

Quando o tráfego de pacotes é muito rápido 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 mesclagem 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 dela 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 a mesclagem de interrupções depende do tamanho do *buffer* de transmissão e recepção. O *buffer* pode ser tanto um espaço de memória da máquina(usando *DMA*, um mecanismo que permite a um dispositivo de E/S usar a memória do sistema como *buffer*) 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 para usar técnicas de mesclagem de interrupções para dispositivos de rede no núcleo do Linux. O objetivo dele 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 *drivers* 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 jogá-los fora 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 acabar gerando 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 ele deve entra ou sair no modo de *polling* e quantos pacotes ele deve pegar 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 mesclagem 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

2.4 Mesclagem de Interrupções na Transmissão

Tanto a transmissão e 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 ser 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].

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 pelo seus limites de hardware e infelizmente continua até hoje. Assim, não é possível enviar pacotes maiores que 1500 *bytes* pela Internet oque 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.

O *GSO*(*Generic segmentation offload*) permite ao *driver* de rede segmentar os pacotes, uma tarefa que normalmente é feita pelo sistema operacional.

Com a segmentação sendo feita fora do sistema, isso permite então enganar o *MTU* na interface de rede do sistema.

Fingindo ter uma *MTU* alta, o pacote é segmentado em pedaços grandes e em menor quantidade quando o sistema manda transmití-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 mesclados quando recebidos. O *LRO* agrega todos os pacotes *TCP* que chegam, mas com possíveis perdas na transformação se por exemplo existe uma diferença nos cabeçalhos no pacote. Já o *GRO* restringe a mesclagem dos pacotes pelos cabeçalhos oque não traz perdas e além disso o *GRO* não é limitado ao *TCP*. Apesar de conseguir uma mesclagem dos pacotes, como já foi dito, a recepção não pode controlar a chegada dos pacotes oque força a ter que usar uma técnica de mesclagem de interrupção como o *NAPI* para conseguir resegmentar os pacotes.

2.5 Mesclagem 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 mesclagem então pode ser feita tanto no *driver* físico como no *driver* virtual de

frontend e *backend*.

[DXZL11] propuseram uma otimização por mesclagem 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 mesclagem de interrupções nas duas causaria um atraso adicional no envio. 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.

2.5.1 Driver do Dispositivo

A mesclagem no *driver* do dispositivo físico é complexa já 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 mesclagem. Quando modificamos o *driver* físico, os requisitos de várias aplicações podem tanto serem satisfeitos como deixarem de ser.

Como exemplo, podemos ter duas aplicação as quais uma requer uma baixa latência e baixa largura de banda, e a outra requer muito processamento e alta largura de banda. Enquanto não mesclar 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. Quando mesclar, a primeira funcionará mal pois a mesclagem irá provocar um atraso considerável na rede e a segunda funcionará bem porque a mesclagem 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.

2.5.2 Netback

A mesclagem no *netback*, diferente do *driver* do dispositivo físico afeta apenas as aplicações de uma determinada máquina virtual. Pelo *netback* estar no domínio do *driver* consumindo processamento junto com vários outros *netbacks*, reduzir as interrupções dele aliviaria o processamento da rede por máquina virtual do domínio do *driver* e poderia permitir que mais máquinas virtuais usem a rede.

Seria necessário analisar os requisitos de aplicação de cada máquina virtual para definir os parâmetros da mesclagem de cada *netback*. Pelo *netback* e *netfront* serem virtuais e desacoplados da lógica do *driver* de rede físico, eles podem usar técnicas de mesclagem se tiverem suporte independente do *driver* de rede físico.

2.5.3 Netfront

A mesclagem no *netfront* como o *netback* depende das aplicações da máquina virtual que a pertence. O ganho pode ser menor em relação ao *netback* já que irá reduzir a interrupção no núcleo da 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 o desempenho da rede em infraestruturas de máquinas virtuais utilizados 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 requerimentos de um usuário.

3.2 Critérios de Seleção

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

Os artigos de relevância alta serão lidos por completo e resumidos. Os artigos médios terão a leitura de sua introdução e a mudança da sua relevância para baixa ou alta. Por fim os artigos baixos não serão lidos.

3.3 Execução

A tabela 3.1 mostra os artigos coletados e sua relevância.

3.3.1 Resumo Sintetizado

Em [CCW+08] o autor fez uma comparação entre o XEN, VMWare e OpenVZ. Nos experimentos foi concluído que o *hypervisor* XEN tem um desempenho baixo em termos de atraso na rede, 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 de não entram em detalhes sobre os motivos dos resultados terem sido esses.

[EF10] estudou 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 foi 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

Tabela 3.1: *Artigos selecionados e suas relevâncias*

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

um gargalo no `dom0`.

[Rix08] fez 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. 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, seria 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 usada a ideia de multi-filas e em adição removeram o domínio de E/S. Sem o responsável por controlar as filas, o *hypervisor* passa a considerar cada conjunto de fila como um interface de rede física e a associa o controlador a uma MV. Assim, cada MV consegue enviar ou receber informações diretamente da rede sem nenhuma intervensão 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*.

[WR12] cita 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.

[Liu10] fez 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 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 mitigação 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 microsegundo em relação a máquina pura. Quando a opção de mitigação 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 todos 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.

[STJP08] propôs modificar a arquitetura do *driver* de E/S do *XEN* para conseguir melhorar o uso da *CPU*. Dentro dos problemas que ele encontrou está o excesso de cópias de dados, a fragmentação de pacotes no *socket*, a falta de alinhamento do cache e o filtro de rede da ponte. Com algumas modificações ele conseguiu uma economia de 56% no uso do processador.

[ON09] analisou o desempenho de um sistema virtualizado com *XEN* aplicando a estratégia *LRO(large receive offload)* onde ainda dentro do *driver* da placa de rede é recebido e reunido os pacotes de informações que tiveram que ser segmentados. Nesse experimento eles mediram 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.

[xen12] sugere algumas práticas para tentar evitar um baixo desempenho em aplicações intensivas de rede/disco no *XEN* como dedicar exclusivamente um núcleo de processador ou uma quantidade de memória para o `dom0`.

[LGBK08] propôs duas otimizações na virtualização de rede: o escalonamento ciente de cache 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 ganho de 96% na largura de banda.

[AMN06] pesquisaram 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 nas duas, tanto a transmissão como a recepção de pacotes perderam mais de 50% do desempenho comparado com a máquina física. Também foi notado que ao rodar o `domU` e o `dom0` em *CPUs* distintas é mais custoso que rodar elas juntas na mesma *CPU*.

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

[FA12] fizeram experimentos em torno do problema da carga extra na virtualização da rede. Para isso eles propuseram adequar o balanço de interrupções para demostrar a possibilidade de reduzir o número de pacotes perdidos. O resultado foi que um balanço 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, no final, discutiram a possibilidade de usar a função de mesclagem existentes nos *drivers* das placas de rede modernas.

[DXZL11] propuseram otimizações para reduzir a carga extra na virtualização da rede. Uma das otimizações foi mesclar eficientemente as interrupções virtuais e a outra escalar o lado da recepção. A mesclagem 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 mesclagem 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 multi-núcleo. Assim, foi introduzido o conceito de *RSS* que balanceia a carga de trabalho eficientemente entre os processadores.

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

3.3.2 Análise das Informações

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 na correlação entre os artigos.

Nos experimentos todos parecem terem feito medições em infraestruturas reais usando o *hypervisor XEN* e alguma distribuição *Linux* como sistema operacional. Isso talvez ocorreu por eles terem o código aberto e o XEN suportar diferentes sistemas operacionais.

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.

3.3.3 Conclusão

A revisão ajudou a entender melhor a área de virtualização de rede. Diversas formas de melhorar o desempenho foram encontradas. Numa próxima revisão, seria interessante focar em alguma categoria para facilitar a comparação entre pesquisas.

Capítulo 4

Proposta

4.1 Tema de Pesquisa

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

4.2 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.

4.3 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].

[EF10] fizeram experimentos 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.

4.4 Relevância do problema

Muitas organizações tem 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].

Na revisão bibliográfica foram encontrados vários autores que propuseram técnicas para tentar resolver esse problema. [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*.

4.5 Propostas da Literatura

Os detalhes de cada proposta da literatura podem ser vistos na subseção resumo sintetizado da seção da revisão bibliográfica.

4.6 Proposta de pesquisa

Na revisão 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 mesclagem 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.

Apesar de não garantir que essa estratégia irá ser melhor que as outras, podemos juntar 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 mesclagem de interrupções dos *drivers* das placas de rede virtuais e física tentando ajustar os parâmetros de mesclagem dinamicamente de forma a garantir uma melhor qualidade dos serviços da infraestrutura de nuvem que utilizam a rede. A qualidade a qual estamos focando está em atributos de desempenho, em específico, a latência e o uso do processador. Estes variam quando modificamos os parâmetros de mesclagem.

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 mesclagem pode ser feita tanto no *driver* físico como no *driver* virtual de *frontend* e *backend* como foi dita na seção de mesclagem e virtualização de rede no capítulo de conceitos. Se pensarmos em mesclar 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 mesclar 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* como exemplo, 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 mesclagem de interrupções na transmissão não foram encontrados experimentos, mas [CRKH05] mostra que a quantidade de interrupção devido a transmissão de pacotes é equivalente as interrupções de recepção na virtualização de rede do *XEN*.

Ainda não foi encontrado nenhum artigo que analisa a mesclagem desses três *drivers* ao mesmo tempo.

Uma análise de como os parâmetros estão relacionados e como eles influenciam no tráfego de rede seria necessário antes de elaborar um algoritmo.

4.7 Questão de pesquisa

O algoritmo de mesclagem 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?

4.8 Cronograma

Revisão bibliografica adicional: Julho/Agosto

Experimento analisando a relação entre cada parâmetro da mesclagem nos *drivers*: Setembro/Outubro

Experimento avaliando o algoritmo proposto: Novembro/Dezembro

Texto da dissertação: Janeiro/Fevereiro

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, 4

[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. 1, 12, 13, 15

[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, 4

[BM99] S. BRADNER e J. MCQUAID. Rfc 2544. *Benchmarking methodology for network interconnect devices*, 1999. 5

[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, 3, 4, 5, 11, 12

[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. 15

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

[Cor09] Jonathan Corbet. Generic receive offload. http://lwn.net/Articles/358910/, 2009. Acessado em: 25/6/2012. 8, 12

[CRKH05] Jonathan Corbet, Alessandro Rubini e Greg Kroah-Hartman. *Linux Device Drivers, 3rd Edition*. O'Reilly Media, Inc., 2005. 7, 8, 16

[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. 4

[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. 7, 9, 12, 14, 15

[Eas07] Thomas M. Eastep. Xen network environment. http://www1.shorewall.net/XenMyWay.html, 2007. Acessado em: 25/6/2012. ix, 5, 6

[EF10] J. Ekanayake e G. Fox. High performance parallel computing with clouds and cloud technologies. *Cloud Computing*, páginas 20–38, 2010. 1, 5, 11, 12, 15

[FA12] T. Fortuna e B. Adamczyk. Improving packet reception and forwarding within virtualized xen environments. *Computer Networks*, páginas 153–160, 2012. 12, 14, 15

[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, 3, 15

[Gee09] Jeremy Geelan. The top 150 players in cloud computing. http://cloudcomputing.sys-con.com/node/770174, 2009. Acessado em: 16/7/2012. 15

[int12] Intel server adapters. http://www.intel.com/support/network/adapter/pro100/sb/CS-031492.htm, 2012. Acessado em: 19/7/2012. 15

[Jam04] T.Y. James. Performance evaluation of linux bridge. Em *Telecommunications System Management Conference*, 2004. 6

[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. 12, 14, 15

[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. 12, 13, 15

[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, 5, 12, 13, 15

[Low09] Scott Low. What is sr-iov? http://blog.scottlowe.org/2009/12/02/what-is-sr-iov/, 2009. Acessado em: 19/7/2012. 15

[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. 8

[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. 12, 13, 15

[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. 4

[Rix08] S. Rixner. Network virtualization: Breaking the performance barrier. *Queue*, 6(1):36–ff, 2008. 1, 4, 5, 12, 15

[Sal07] K. Salah. To coalesce or not to coalesce. *AEU-International Journal of Electronics and Communications*, 61(4):215–225, 2007. 7

[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. 12

[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. 4

[Spe10] Stephen Spector. New to xen guide. http://www.xen.org/files/Marketing/NewtoXenGuide.pdf, 2010. Acessado em: 16/7/2012. 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. ix, 5, 6, 7, 12, 13, 15

[WR12] C. Waldspurger e M. Rosenblum. I/o virtualization. *Communications of the ACM*, 55(1):66–73, 2012. 1, 5, 12, 15

[xen12] Xen best practices. http://wiki.xen.org/wiki/XenBestPractices, 2012. Acessado em: 25/6/2012. 13, 15