

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, agosto de 2012

Agregação de Interrupções de placas de rede para Infraestruturas em Nuvem

Dissertação apresentada ao
Instituto de Matemática e Estatística da Universidade de São Paulo
como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação.

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 gerência de 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 management 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	vii
Lista de Símbolos	ix
Lista de Figuras	xi
Lista de Tabelas	xiii
1 Introdução	1
1.1 Objetivos	2
1.2 Contribuições	2
1.3 Organização do Trabalho	2
2 Conceitos	5
2.1 Arquitetura de E/S em Computadores	5
2.2 Computação em Nuvem	5
2.3 Virtualização	6
2.3.1 Virtualização de Servidores	7
2.3.2 Virtualização de Dispositivos de E/S	8
2.3.3 Virtualização da Rede	8
2.4 Agregação de Interrupções na Recepção	11
2.5 Agregação de Interrupções na Transmissão	12
2.6 Agregação e Virtualização de Rede	12
2.6.1 Driver do Dispositivo	13
2.6.2 Netback	13
2.6.3 Netfront	13
3 Revisão Bibliográfica	15
3.1 Objetivo	15
3.2 Critérios de Seleção	15
3.2.1 Resumo Sintetizado	16
3.2.2 Análise das Informações	18
3.2.3 Resumo Conclusivo	18
4 Proposta	19
4.1 Tema de Pesquisa	19

4.2	Problema de Pesquisa	19
4.3	Evidências do Problema	19
4.4	Relevância do problema	19
4.5	Proposta de Pesquisa	20
4.5.1	Algoritmo	20
4.6	Questão de Pesquisa	21
4.7	Cronograma	21
Referências Bibliográficas		23

Lista de Abreviaturas

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

Lista de Símbolos

Lista de Figuras

1.1	taxa de transferência para diferentes dispositivos de E/S [Sta10]	2
2.1	três técnicas para operações de E/S [Sta10]	6
2.2	ponte virtual criada no XEN [Eas07]	10
2.3	roteador virtual criado no XEN [rou10]	10
2.4	arquitetura da rede virtual no XEN [STJP08]	11

Lista de Tabelas

2.1	Parâmetros para agregação de interrupções	12
3.1	Artigos selecionados e suas relevâncias	15

Capítulo 1

Introdução

Em infraestruturas de nuvem, recursos computacionais são controlados pelo fornecedor e alocados a qualquer momento de acordo com a necessidade do consumidor [GED⁺11]. Para obter essa funcionalidade de alocar recursos a qualquer momento, normalmente, 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 computacionais 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 usar uma estratégia para gerenciar a agregação de interrupções. A agregação de interrupções faz o *driver* da placa de rede não gerar 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 é gerada quando uma quantidade de pacotes é transmitida/recebida. Essa pesquisa foca em propor algoritmos que automatizem a configuração de agregação de interrupções realizada pelo *hypervisor*, tentando ajustar esse parâmetro dinamicamente de forma a garantir os requisitos de Qualidade de Serviço (QoS) de aplicações em nuvem que façam uso intensivo dos enlaces de rede.~~

Para reduzir o custo do processamento, uma proposta é usar 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 é gerada quando uma quantidade de pacotes é transmitida/recebida, reduzindo então, a quantidade de interrupções por pacote. As interrupções têm prioridade em serem realizadas pelo sistema em relação a outras tarefas. Considerando os dispositivos de E/S, as redes gigabits transferem com um taxa 10x maior que os discos rígidos e 1000x maior que os disquetes como é possível ver na Figura 1.1 [Sta10].

Quando o sistema é sobrecarregado de interrupções, este gasta todo seu tempo tentando resolvê-la, impossibilitando que outras tarefas sejam executadas. Essa situação é chamada de “*livelock*” [Sal07]. Na recepção de pacotes pode ocorrer um *livelock* quando a transferência de pacotes é em

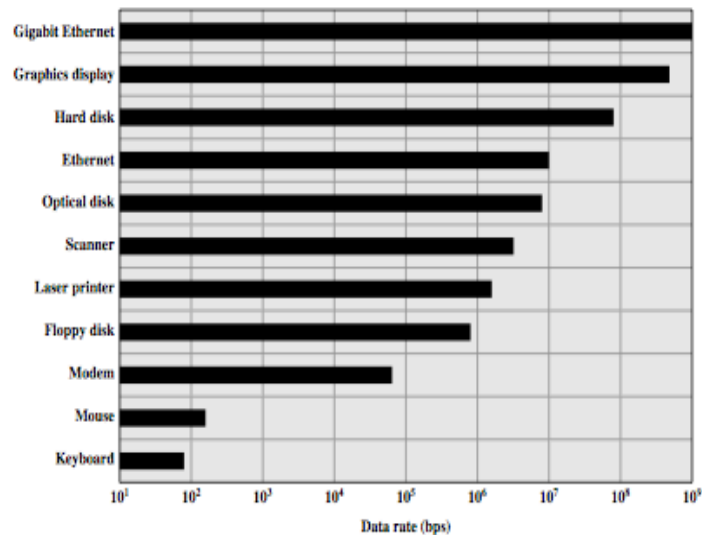


Figure 2.11 Typical I/O Device Data Rates

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

torno de gigabits e vários pacotes são recebidos enquanto um outro ainda está sendo processado. Nesse cenário, o sistema passa a ter que manipular cada uma das interrupções geradas pelos pacotes que chegam e nunca consegue processar um pacote ou então começa a descartar pacotes.

Essa pesquisa foca em propor algoritmos que automatizem a configuração de agregação de interrupções realizada pelo *hypervisor*. Esses algoritmos tentam ajustar os parâmetro de agregação de interrupções dinamicamente de forma a garantir os requisitos de Qualidade de Serviço (*QoS*) de aplicações em nuvem que façam uso intensivo dos enlaces de rede.

1.1 Objetivos

O objetivo dessa pesquisa é demonstrar que os algoritmos de agregação de interrupções reduzem o uso da *CPU* por pacote na transmissão e recepção e mantêm a latência consideravelmente baixa, dentro dos requisitos das aplicações. O objetivo dessa pesquisa é demonstrar que os algoritmos de agregação de interrupções propostos reduzem o uso da *CPU* por pacote na transmissão e recepção e mantêm 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 4 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

2.1 Arquitetura de E/S em Computadores

A arquitetura de E/S dos computadores é controlada por um módulo específico para E/S. Este módulo de E/S é necessário para o processador conseguir se comunicar com um ou mais dispositivos externos de E/S. Cada dispositivo possui vários métodos de operação, diferentes formatos, comprimento de palavras e velocidade de transferência, o que faz cada módulo ter uma lógica específica para um determinado 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.

Existem três técnicas possíveis para operações de E/S: E/S programada, interrupção dirigida à E/S e *DMA* (acesso direto a memória). Com a 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. 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. No *DMA* um processador especializado em E/S toma controle das operações de E/S para mover um grande bloco de dados usando a memória principal. Na Figura 2.1 vemos as três técnicas sendo aplicadas para receber um bloco de dados de um dispositivo de E/S.

A interrupção dirigida à E/S resolve o problema da E/S programada em que o processador tem que esperar um longo tempo para o módulo de E/S estar pronto para recepção ou transmissão dos dados prejudicando o sistema. Um problema da interrupção dirigida e da E/S programada é que ambos usam o processador, o que pode adicionar uma carga extra no processador. No *DMA*, o módulo tem acesso direto a memória, podendo agora, mover blocos de dados para ou da memória sem envolver a *CPU*, excetuando no começo e fim da transferência.

2.2 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 nuvem é muito abrangente, ele foi dividido em várias classificações [AFG⁺09], entre elas, o tipo de serviço o qual fornecem. Seguindo essa classificação existem as nuvens que fornecem *softwares* como serviço (SaaS – *Software As A Service*), plataformas como serviço (PaaS – *Platform As A Service*), e infraestruturas como serviços (IaaS – *Infrastructure as a Service*). Nuvens que fornecem *SaaS* são utilizadas por exemplo pelo *Google Docs*¹ que fornece um *software* para edição de documentos como serviço. Nuvens que fornecem PaaS são utilizadas em casos como o *Google App Engine*² e o *Windows Azure*³ onde é oferecido uma plataforma para implantar apli-

¹<http://docs.google.com>

²<http://developers.google.com/appengine/>

³<http://www.windowsazure.com/en-us/>

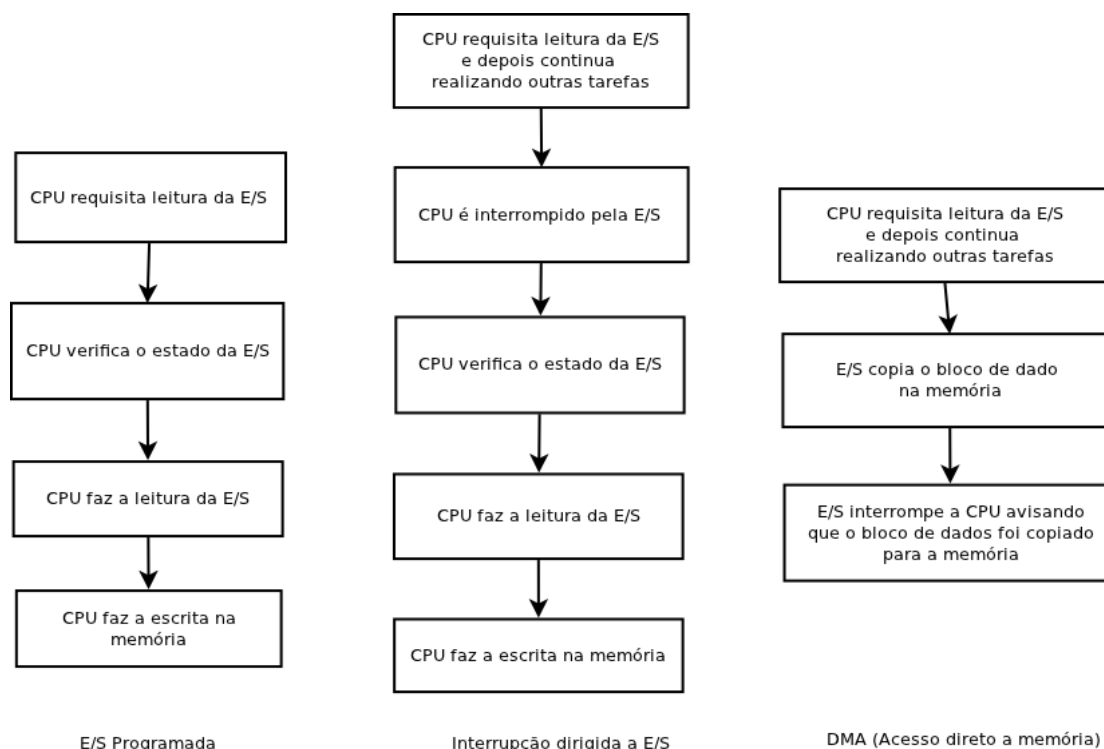


Figura 2.1: três técnicas para operações de E/S [Sta10]

cações e nuvens que fornecem *IaaS* são utilizadas em casos como o *Amazon EC2*⁴ que fornece uma infraestrutura que emula um computador.

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 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 dentro de um nó da infraestrutura.

2.3 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]. 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, a única máquina afetada seria a máquina virtual utilizada por aquele programador, já que apenas uma parte do processamento total é reservada e não é possível expandir a quantidade dos recursos sem permissão do administrador da nuvem.

Além da segurança, outra consequência da virtualização é que ela 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

⁴<http://aws.amazon.com/ec2/>

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.

2.3.1 Virtualização de Servidores

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 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 *hypervisors* estão o *KVM*¹, o *XEN*², o *VMWare*³ e o *VirtualBox*⁴.

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. Na para-virtualização, 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 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. Isto as tornam mais simples e mais portáteis 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, 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

Com a virtualização de servidores, os dispositivos de E/S físicos passam a ter que sofrer modificações já que em um servidor não há 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, 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, *hypervisors* 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] faz algumas menções sobre o uso de técnicas de virtualização de E/S que desacoplam a implementação física do dispositivo lógico. 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, 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 [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.

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.

[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 2.2. 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* é configurado para encaminhar pacotes *IP* e os *domUs* levantam uma interface virtual *vifX.0* para se comunicar com o *dom0*. A figura 2.3 mostra um exemplo de configuração do tipo roteador.

[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 [CCW⁺08], [EF10], [WR12], [SBB⁺07], [STJP08], [ON09], [Cor09], [LGBK08], [AMN06], [JSJK11], [FA12], [DXZL11] terem feito experimentos com essa configuração.

Na Figura 2.4 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 (*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 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 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.

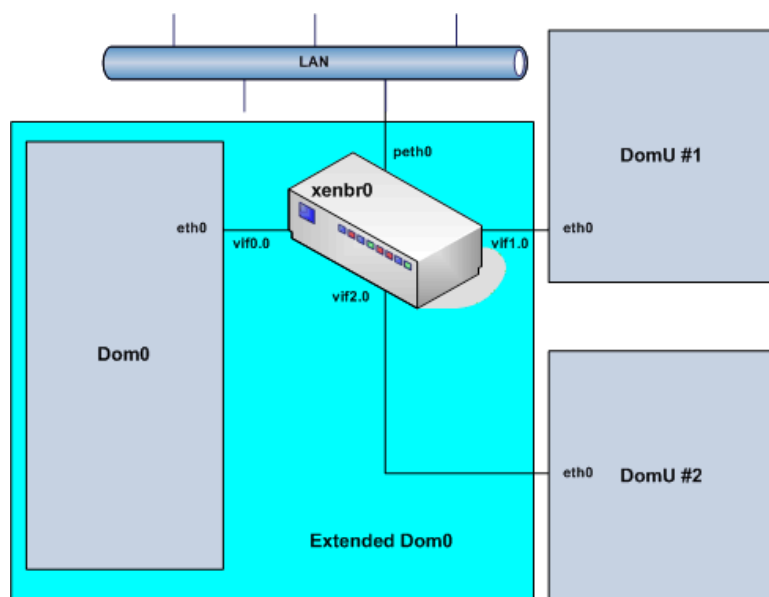


Figura 2.2: ponte virtual criada no XEN [Eas07]

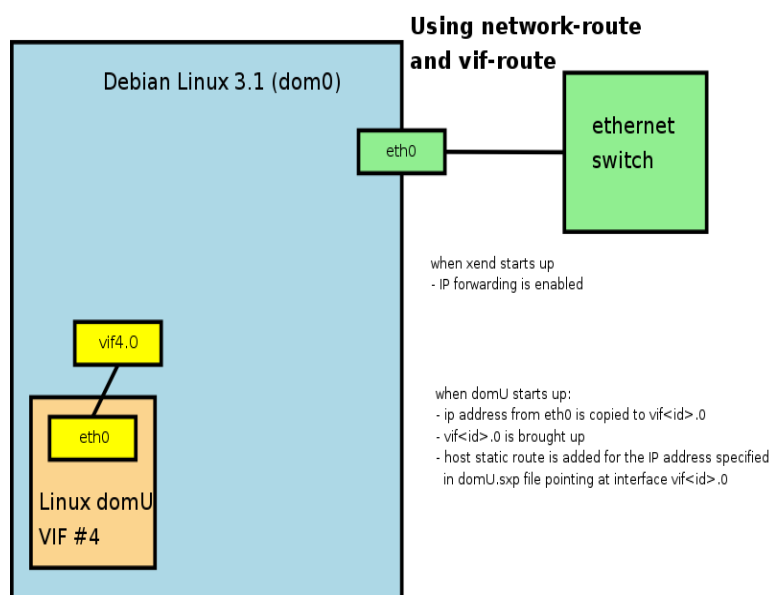


Figura 2.3: roteador virtual criado no XEN [rou10]

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 à ponte dentro do domínio do *driver*. 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 notificação para o mesmo. Quando o *netfront* recebe a notificação, 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].

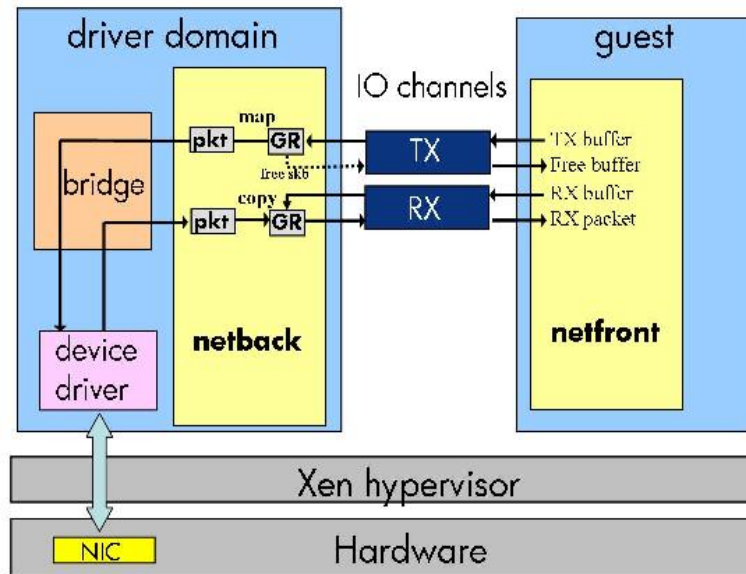


Figura 2.4: arquitetura da rede virtual no XEN [STJP08]

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 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 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 (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 agregação 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 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 entrar 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 sistema operacional.

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 fora do sistema, pode-se “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 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 agregados 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 agregação dos pacotes pelos cabeçalhos, o que não traz perdas e além disso o *GRO* não é limitado ao *TCP*. Apesar de conseguir uma agregação dos pacotes, como já foi dito, a recepção não pode controlar a chegada dos pacotes o que força a utilização de uma técnica de agregação de interrupção 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*.

[DXZL11] propuseram 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 já que afeta o tráfego de pacotes em todas máquinas virtuais e, conseqüentemente, em todas as aplicações que usam a rede. Também necessita que a placa de rede tenha suporte a agregação. 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 agregar 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 agregar, 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 Netback

A agregação 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 usassem a rede.

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

2.6.3 Netfront

A agregação no *netfront* depende somente das aplicações da máquina controladora do *netfront*. O ganho pode ser menor em relação ao *netback* 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 pela estratégia de busca 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-baixa
[AMN06]	alta
[JSJK11]	alta
[FA12]	alta
[DXZL11]	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, 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.

[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 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 criada 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, 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*.

[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 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 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 microssegundo 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 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.

[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 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.

[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 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 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 proces-

sador e 8% de ganho de largura 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 demonstrar 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 agregação existentes nos *drivers* das placas de rede modernas.

[DXZL11] propuseram 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 multi-núcleo. 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 foi até 2,2 vezes mais, também na largura de banda.

3.2.2 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 a correlação entre os artigos.

Nos experimentos todos parecem ter 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.

3.2.3 Resumo Conclusivo

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

Proposta

4.1 Tema de Pesquisa

Essa pesquisa tem como tema técnicas de otimização na da 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]. 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].

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]. Muitas organizações tem 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*.

4.5 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, conseqüentemente, poderia melhorar o desempenho da infraestrutura de nuvem.

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 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 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 2.5. 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 [CRKH05] mostra que a quantidade de interrupções devido a transmissão de pacotes é equivalente às interrupções de recepção na virtualização de rede do *XEN*.

Ainda não foi encontrado 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.

4.5.1 Algoritmo

Na primeira proposta de algoritmo, separaremos as aplicações que estão dentro da nuvem em classes de serviços e de acordo com os dados obtidos nessa classificação, iremos variar os parâmetros de agregação de interrupções. A classe de serviços é dividida em A, B e C. Na classe A a aplicação exige a menor latência possível, na classe B a aplicação exige uma latência de *X* ms. Enquanto que a classe C não exige latência.

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 baseados no requisito de aplicação que exigir a menor latência

4. Os parâmetros de agregação da placa de rede virtual de *backend* e *frontend* serão baseados no requisito da aplicação dentro da máquina virtual que exigir a menor latência.

4.6 Questão de Pesquisa

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

4.7 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

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, 5, 7
- [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, 19
- [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 Comunicação-XXIX CSBC-Congresso da Sociedade Brasileira de Computação*, 2009. 8
- [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, 7
- [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, 6, 7, 8, 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. 19
- [cor05] corbet. Napi performance - a weighty matter. <http://lwn.net/Articles/139884/>, 2005. Acessado em: 16/7/2012. 12
- [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, 12, 20
- [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. 7
- [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, 13, 15, 18, 19

- [Eas07] Thomas M. Eastep. Xen network environment. <http://www1.shorewall.net/XenMyWay.html>, 2007. Acessado em: 25/6/2012. xi, 9, 10
- [EF10] J. Ekanayake e G. Fox. High performance parallel computing with clouds and cloud technologies. *Cloud Computing*, páginas 20–38, 2010. 1, 8, 9, 15, 16, 19
- [FA12] T. Fortuna e B. Adamczyk. Improving packet reception and forwarding within virtualized xen environments. *Computer Networks*, páginas 153–160, 2012. 9, 15, 18, 19
- [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, 8, 19
- [Gee09] Jeremy Geelan. The top 150 players in cloud computing. <http://cloudcomputing.sys-con.com/node/770174>, 2009. Acessado em: 16/7/2012. 19
- [int12] Intel server adapters. <http://www.intel.com/support/network/adapter/pro100/sb/CS-031492.htm>, 2012. Acessado em: 19/7/2012. 19
- [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, 19
- [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, 19
- [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, 8, 15, 16, 19
- [Low09] Scott Low. What is sr-iov? <http://blog.scottlowe.org/2009/12/02/what-is-sr-iov/>, 2009. Acessado em: 19/7/2012. 19
- [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. 12
- [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, 19
- [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. 7
- [Rix08] S. Rixner. Network virtualization: Breaking the performance barrier. *Queue*, 6(1):36–ff, 2008. 1, 8, 15, 16, 19
- [rou10] Xen networking. http://wiki.xen.org/wiki/Xen_Networking, 2010. Acessado em: 16/7/2012. xi, 10
- [Sal07] K. Salah. To coalesce or not to coalesce. *AEU-International Journal of Electronics and Communications*, 61(4):215–225, 2007. 1, 11

- [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. Bouffleur, R.C.M. dos Santos e A.S. Charao. Análise de desempenho da virtualizaç ao de rede nos sistemas xen e openvz. 7
- [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, 6
- [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, 8, 9, 10, 11, 15, 17, 19
- [WR12] C. Waldspurger e M. Rosenblum. I/o virtualization. *Communications of the ACM*, 55(1):66–73, 2012. 1, 8, 9, 15, 16, 19
- [xen12] Xen best practices. <http://wiki.xen.org/wiki/XenBestPractices>, 2012. Acessado em: 25/6/2012. 17, 19