# Avaliação de ambientes virtualizados através de um sistema de distribuição de vídeo sob demanda

Diego Dutra
LCP/PESC/COPPE
UFRJ
diegodutra@lcp.coppe.ufrj.br

Lauro Whately
LCP/PESC/COPPE
UFRJ
whately@lcp.coppe.ufrj.br

Claudio Amorim
LCP/PESC/COPPE
UFRJ
amorim@cos.ufrj.br

27 de junho de 2011

**Resumo**

Ambientes virtualizados tem se tornado a plataforma básica para desenvolvimento de serviços Web nos últimos anos. Esta tendência no projeto e construção de serviços Web decorre em parte das garantias de isolamento fornecidas pelas máquinas virtuais e pelo surgimento de ambientes de computação em nuvem conhecidos como infraestrutura como serviço (em inglês, *Infrastructure as a Service* - **IaaS**).

A construção de um serviço de distribuição de vídeo sob demanda (**VsD**) através da Internet de maneira escalável apresenta uma serie de complexidades, o que faz com que os sistemas existentes sejam projetados de forma ad hoc. Uma das forma que têm sido utilizada para construir serviços de VsD escaláveis é através do uso de clusters de computadores, de forma que os clientes passam a receber fluxos de vídeo de um dos nós destes clusters. Neste sentido faz-se necessário o entendimento do sistema computacional sobre o qual executa o serviço de VsD, de maneira a permitir aos projetistas dos mesmos utilizarem de forma mais eficiente os recursos computacionais disponíveis. Uma extensão no projeto utilizando clusters dedicados é o uso de clusters de computadores virtualizados, sejam estes privados ou em um provedor de IaaS como a Amazon. Contudo o acréscimo da virtualização nos sistemas computacionais que suportam o serviço de VsD deve ser levado em consideração no projeto dos mesmos.

Esta trabalho tem como objetivo avaliar o impacto que diferentes técnicas de virtualização tem sobre um sistema de distribuição de vídeo sob demanda, para isso é são realizados experimentos de como um servidor HTTP que suporta o serviço se comporta quando executado dentro de um ambiente virtualizado. Tendo sido realizada uma analise utilizando três técnicas de virtualização distintas representadas pelos softwares KVM, Xen e OpenVZ.

# Capítulo 1

# Introdução

A construção de um serviço de distribuição de vídeo sob demanda (**VsD**) através da Internet de maneira escalável apresenta uma serie de complexidades, o que faz com que os sistemas existentes sejam projetados de forma ad hoc. Uma das maneiras encontradas para a construção de tais sistemas tem sido através do uso de clusters de computadores, de forma que os clientes passam a receber fluxos de vídeo de um dos nós destes clusters. Contudo essas soluções comumente levam em consideração que o serviço de distribuição executa diretamente sobre os nós físicos do cluster e não compartilham os recursos físicos com mais nenhuma aplicação.

Contudo é possível observar que ambientes virtualizados tem se tornado a plataforma básica para desenvolvimento de serviços Web nos últimos anos. Esta tendência no projeto e construção de serviços Web decorre em parte das garantias de isolamento fornecidas pelas máquinas virtuais e pelo surgimento de ambientes de computação em nuvem conhecidos como infraestrutura como serviço (em inglês, *Infrastructure as a Service* - **IaaS**). Em especial, Whatley [1] demonstrou a viabilidade de se construir um serviço de VsD sobre um cluster com nós virtualizados e garantir com isso a qualidade de serviço (QoS) do sistema de distribuição.

Contudo faz-se necessário ainda o entendimento do sistema computacional sobre o qual executa o serviço de VsD, de maneira a permitir aos projetistas dos mesmos utilizarem de forma mais eficiente os recursos computacionais disponíveis. Deste modo este trabalho tem como objetivo avaliar o impacto que diferentes técnicas de virtualização tem sobre um sistema de distribuição de vídeo sob demanda. Demonstrando experimentalmente como as diferentes técnicas podem impactar no desempenho do servidor de distribuição com relação ao uso dos recursos computacionais disponíveis.

## 1.1 Motivação

O compartilhamento dos recursos computacionais levanta questões sobre o isolamento das aplicações que executam sobre estes. Uma forma que tem sido utilizada para oferecer algumas garantias de isolamento entre as diversas aplicações é o uso de servidores virtualizados. As técnicas de virtualização oferecem diferentes graus de isolamento entre as máquinas virtuais em troca de um possível

aumento do *overhead* ou necessidade do uso de *hardware* com suporte especial. Sendo importante entender como tais características impactam no serviço de distribuição de vídeos.

## 1.2 Objetivo

Avaliar quantitativamente o impacto que diferentes técnicas de virtualização tem sobre um serviço Web para a distribuição de vídeo sob demanda escalável.

## 1.3 Organização do texto

Este relatório apresenta no Capítulo 2 os conceitos básicos, e serviços Web escaláveis, necessários para o entendimento deste trabalho, entre eles os de máquina virtual, computação na nuven com uma introdução aos conceitos de elasticidade e resiliência para estes ambientes computacionais, o sistema de vídeo sob demanda GloVE e o sistema de transmissão de vídeos sob demanda SMART. No Capítulo 3, são apresentados os principais trabalhos relacionados com o conteúdo deste trabalho. O Capítulo 4 descreve o serviço de distribuição de vídeo sob demanda considerado neste trabalho, enquanto o Capítulo 5 apresenta alguns resultados experimentais. Por fim, no Capítulo 6, é feita a conclusão deste relatório técnico.

# Capítulo 2

# Conceitos Básicos

Este capítulo tem como objetivo revisar conceitos básicos, definições e questões envolvidas neste trabalho. Ele também apresenta o modelo de um sistema de distribuição de vídeo sob demanda e os questionamentos arquiteturais que motivaram, este trabalho. Após, são feitas algumas considerações finais, encadeando os conteúdos abordados de forma a delimitar o escopo da pesquisa.

## 2.1 Máquinas Virtuais e Técnicas de Virtualização

As tecnologias de virtualização encontram-se difundidas em diversos níveis dos sistemas digitais. Nos últimos anos, o uso de máquinas virtuais ganhou grande impulso devido à disponibilidade cada vez maior de poder computacional dos processadores com múltiplos núcleos e com o suporte, nestes novos processadores, para tecnologias de virtualização como as arquiteturas Intel VT [2, 3] e *AMD Virtualization* (**Pacifica**).

Dentro do escopo deste trabalho, máquinas virtuais e técnicas de virtualização aparecem como uma principal motivação do problema estudado. A razão é que, dentro do escopo deste trabalho os recursos computacionais disponíveis são organizados como máquinas virtuais e disponibilizados aos usuários do sistema computacional. Assim sendo, é importante entender esses mecanismos, como eles podem ser úteis na construção de sistemas computacionais e como afetam as aplicações de transmissão de vídeo e científicas.

### 2.1.1 Conceitos Gerais

Um dos conceitos básicos em computação é o uso de abstrações para esconder do desenvolvedor as complexidades do *hardware*. O uso de múltiplos níveis de abstração pretende criar, em cada nível, máquinas virtuais que estendem as funcionalidades disponíveis no nível abaixo, construindo desta forma uma hierarquia de abstrações [4].

Além das funcionalidades estendidas, uma máquina virtual deve criar isomorfismos entre as funcionalidades que ela oferece ao nível acima e o que pode ser executado na máquina virtual do nível abaixo. Dentro do escopo deste relatório, virtualização é definida como a construção de um mapeamento entre o

dispositivo virtual (funcionalidade) e o real [5].

### 2.1.2 Virtualização Total

A classe de Máquinas Virtuais que implementam a Virtualização Total oferece para as aplicações que executam sobre essa máquina virtual um *hardware* idêntico ao que é fornecido pela máquina física. Nela, existe uma camada de *software* ou *firmware*, que é responsável por gerenciar os recursos físicos e exportar para as camadas acima (SO e aplicações) uma interface igual a do *hardware* [6, 7]. Este modelo de virtualização pode ser implementado com auxílio da arquitetura do processador ou via tradução binária [7].

#### Kernel-based Virtual Machine

Com a chegada no mercado de processadores da arquitetura x86 que possuem suporte em hardware para a virtualização [2, 3], foi implementado no *kernel* do Linux um MMV, o ***K**ernel-based **V**irtual **M**achine* (KVM) [8]. O uso das extensões de virtualização dos novos processadores permite que uma instrução que viola as regras impostas pelos mecanismos de virtualização sejam executadas, sendo que a execução destas instruções causa uma interrupção que deve ser tratada pelo monitor. O **KVM** é um subsistema do Linux que se utiliza do suporte do *hardware* disponível nestes processadores para construir um monitor de máquina virtual. Sendo que, inicialmente o suporte de entrada/saída (E/S) era realizado através **QEMU** [9].

O KVM é implementado como um módulo no *kernel* do Linux tradicional, ele realiza a execução de uma máquina virtual através da criação de duas *threads* dentro do kernel e uma instância em modo usuário do QEMU. Uma destas *threads* representa o fluxo de execução da máquina virtual enquanto a outra é utilizada para fazer a interface com o QEMU executando em modo usuário.

A Figura 2.1 mostra uma representação de um sistema computacional utilizando o KVM como mecanismo de virtualização. Pode-se observar os três níveis ou modos de trabalho no processador com suporte a virtualização total. Ainda na Figura 2.1 os retângulos em modo usuário representam aplicações dos usuários do sistema computacional, o grande retângulo em modo núcleo representa o *kernel* do sistema operacional (ex. Linux), onde os paralelogramos subsistemas ou módulos dentro do *kernel* e o retângulo em modo virtual representa uma máquina virtual executando. Os processadores que oferecem este suporte implementam um novo modo de trabalho, o modo Virtual, onde as máquinas virtuais podem executar nativamente e quando executam alguma instrução que necessite de intervenção do MMV, provoca uma interrupção no sistema computacional e o controle do núcleo de processamento é entregue ao MMV. Um exemplo de instruções que não exige a intervenção do MMV são as instruções ligadas as chamadas de E/S da máquina virtual.

Ainda na Figura 2.1, as setas numeradas marcam essas chamadas, a seta 1 representa a comunicação entre uma aplicação rodando dentro de uma máquina virtual e o subsistema de E/S do *kernel* do Linux que executa dentro da máquina virtual, por sua vez esse necessita interagir como o modulo do KVM (seta 2), que repassa a chamada de E/S para a aplicação QEMU (seta 3), que encontra-se executando no nível do usuário, o QEMU por sua vez encaminha a chamada

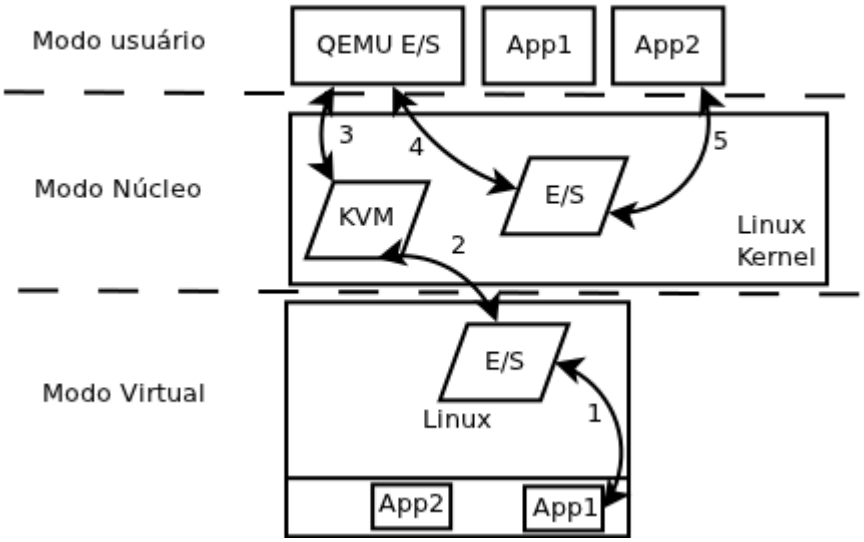


Figura 2.1: Arquitetura do **KVM**

ao subsistema de E/S do *kernel* do sistema operacional no nó fisico (seta 4). A seta de número 5 representa um pedido de E/S realizado por uma aplicação qualquer que executa no nível do usuário.

### 2.1.3 Paravirtualização

As máquinas virtuais desta classe executam sobre um MMV que é responsável por gerenciar os acessos feitos pelas máquinas virtuais. O MMV permite a construção de máquinas virtuais que são semelhantes à máquina física que ele virtualiza, porém nesta classe de Máquinas Virtuais o conjunto de instruções original não se encontra todo virtualizado, o que obriga as máquinas virtuais, e por consequência o SO que executa sobre estas, a executarem estas instruções não suportadas através de chamadas ao MMV [7].

#### Xen

O **Xen** [10, 11] é um monitor de máquina virtual de código livre que implementa o modelo de paravirtualização, desta forma o núcleo do sistema operacional deve ser modificado de forma a utilizar as chamadas que substituem os recursos que não são virtualizados. Entre estes recursos estão algumas instruções que acessam estados internos do processador sem com isso causar uma interrupção. A implementação do Xen consegue fornecer um mecanismo de virtualização de baixo custo computacional para aplicações que sejam dependentes apenas de CPU.

A Figura 2.2, ilustra como a arquitetura de um sistema computacional usando Xen; pode-se observar o Monitor de Máquina Virtual (**MMV**) e o

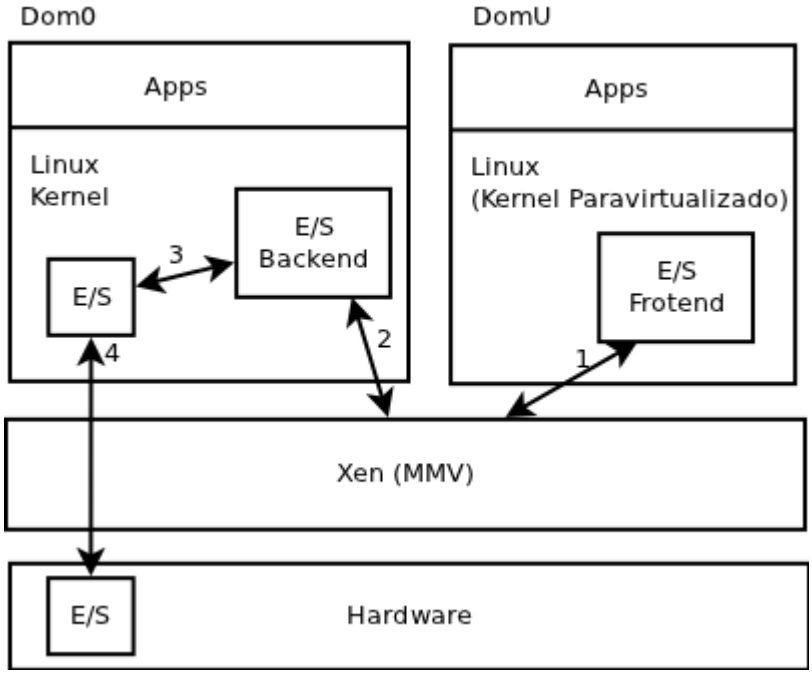


Figura 2.2: Arquitetura do **Xen**

Domínio Privilegiado (**Dom0**) responsável por realizar o acesso aos dispositivos de E/S no *hardware*. Por fim, nesta arquitetura, as máquinas virtuais, ou Domínio não Privilegiado (**DomU**), somente realizam acessos aos dispositivos de E/S através do **Dom0**. Ainda na Figura 2.2, as setas numeradas apresentam o caminho realizado para o atendimento de requisições de E/S, em especial, o driver de E/S da máquina virtual (E/S *Front-End*) comunica-se com o **MMV** (1), o MMV repassa (2) essa requisição para o driver de E/S no Dom0 (E/S *BackEnd*), o *device driver* no Dom0 repassa o pedido (3) ao subsistema de E/S do *kernel*, para finalmente o subsistema de E/S realizar o acesso ao *hardware* (4). Com relação as etapas que ocorrem entre o Dom0, **DomU** e o MMV elas são realizadas através do canal de eventos do Xen e páginas de memória compartilhadas. O mecanismo que controla o precedimento descrito anteriormente no Xen é a *grant table*.

### 2.1.4 Virtualização no Nível do Sistema Operacional

A Virtualização no Nível do Sistema Operacional constrói uma nova abstração dentro dos Sistemas Operacionais tradicionais, os contêineres de recursos (*resource containers*) ou simplesmente contêineres [12, 13]. Um contêiner isola dentro do núcleo do SO todas as estruturas de controle referentes às aplicações que ele encapsula, viabilizando desta forma a utilização de políticas diferentes para o gerenciamento de cada conjunto de aplicações e permitindo um melhor balanceamento de recursos computacionais entre os grupos isolados [14].

Uma das vantagens desta classe de virtualização é o baixo custo imposto

por suas implementações, sendo esse o motivo de sua escolha para arquiteturas compartilhadas como o PlanetLab [15], onde a demanda por capacidade de processamento é alta. O baixo custo computacional (consumo de CPU e memória) das implementações de contêineres é devido a ausência da necessidade de prover mecanismos para interpretação de instruções e outras complexidades necessárias para a virtualização total ou paravirtualização.

### OpenVZ

O **OpenVZ** [16] é um dos diversos trabalhos [12, 13] na literatura que implementam essa classe de virtualização. Um diferencial com relação a outros trabalhos é que no OpenVZ apenas o núcleo do Linux é compartilhado entre os diversos contêineres. Assim, diversos Sistemas Operacionais (aplicativos de sistemas e bibliotecas) podem executar sobre o mesmo núcleo Linux compartilhado de forma transparente.

Com relação à tecnologia de migração, o OpenVZ utiliza um algoritmo de suspender/resumir, onde se faz necessário serializar em um arquivo todo o estado do contêiner antes que esse possa ser recriado no nó computacional de destino. O uso deste tipo de migração faz com que o *downtime* percebido pelas aplicações que executam dentro do contêiner que esta sendo migrado seja costumeiramente superior ao de esquemas de pré-copia.

### Linux Containers

O projeto de *Linux Containers* (**LXC**), oferece o suporte à virtualização de contêineres no Linux. O suporte à esta classe de virtualização no *kernel* oficial do Linux (*Kernel Vanilla*) é feito através de espaços de nomes (*namespaces*) [17], sendo que a introdução do primeiro deles foi na versão 2.6.15 como pode ser visto na Tabela 2.1. A implementação do **LXC** foi feita através de alterações no *kernel* e aplicativos de gerenciamento no nível do usuário, aplicativos esses que utilizam as novas características construídas no *kernel* para suportar contêineres.

Tabela 2.1: Implementação do espaço de nomes usados pelo LXC no *kernel* do Linux ao longo de diferentes versões

| Subsistema | Versão do Kernel |
|---|---|
| Shared SubTrees | 2.6.15 |
| UTSNAME | 2.6.19 |
| PID | 2.6.24 |
| IPC | 2.6.19 |
| USER | 2.6.23 |
| NETWORK | 2.6.26 |
| /PROC | 2.6.26 |
| RO BIND MOUNT | 2.6.24 |

O LXC permite que tanto se isole apenas aplicativos, como todo um sistema operacional de maneira similar ao **OpenVZ** (ver seção 2.1.4). No Linux os espaços de nomes dos recursos são implementados por processos e sendo que cada processo referência um conjunto de espaços de nomes, normalmente compartilhados com outros processos de uma mesmo contêiner [18]. O uso de

espaços provê tanto isolamento quanto virtualização, uma vez que com processos em contêineres diferentes podem possuir recursos com o mesmo **ID**, como por exemplo, dois processos podem deter o mesmo **PID** desde que estejam em contêineres distintos.

A implementação de contêineres no *kernel* oficial do Linux, através do LXC, permitiu o surgimento de um grupo de trabalho que busca construir um mecanismo de migração, ou *checkpoint-restart* (**Linux-CR**) [19], para este sistema operacional. O projeto **Linux-CR** é baseado em experiências de projetos similares que não foram introduzidos no *kernel* [12, 16].

## 2.2 Serviços de computação na nuvem

De forma geral, o termo *cloud computing* [20, 21, 22], ou computação na nuvem, tem sido empregado para designar serviços computacionais em larga escala distribuídos, dinamicamente reconfiguráveis e escaláveis, entregues sob demanda aos usuários e acessíveis através da Internet. Infelizmente, essa definição inicial abre uma gama de opções o que a torna confusa e dificulta o uso do termo. A Figura 2.3 apresenta os diferentes tipos de serviços que são chamados hoje de *cloud computing*. Como estes serviços podem designar componentes em diferentes níveis da arquitetura de um sistema computacional, faz-se necessário para este trabalho definir tanto quais são essas abordagens que hoje são encontradas na literatura e, em especial, qual delas é assumida no restante do texto.

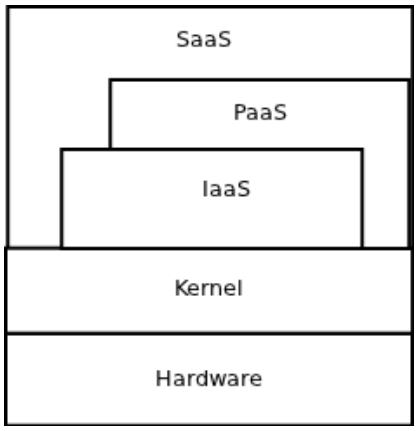


Figura 2.3: Hierarquia de serviços de computação na nuvem

### 2.2.1 Software como serviço - SaaS

Software como serviço (*Software as a Service* - SaaS) [21, 23] é um modelo de distribuição onde aplicativos de propósito específicos são armazenados no provedor de serviço e disponibilizado aos usuários através de uma rede de interconexão como a Internet. Na Figura 2.3, ele fica no topo da hierarquia apresentada, podendo ser construído sobre a hardware direto, sobre um provedor de **PaaS** ou

sobre um provedor de **IaaS**. Como exemplos deste tipo de provedor de serviço são as empresas Salesforce.com [24] e Google.

### 2.2.2 Plataforma como serviço - PaaS

Plataforma como serviço (*Platform as a Service* - PaaS) [21, 25] é um modelo de distribuição de um ambiente integrado de alto nível para construir, testar e disponibilizar aplicativos customizados. Na Figura 2.3, ele fica no meio da hierarquia apresentada, podendo ser construído sobre a *hardware* direto ou sobre um provedor de **IaaS**. O desenvolvimento de aplicações sobre um **PaaS** oferece algumas vantagens para os desenvolvedores (usuários da plataforma), como escalabilidade intrínseca da aplicação e atualização do sistema operacional sem a necessidade de intervenção dos desenvolvedores, sendo necessário porém que os desenvolvedores aceitem algumas restrições de como eles podem escrever seus aplicativos. Este tipo de serviço é provido pela Google em seu **App Engine** [26] e pelo sistema de código aberto **AppScale** [27].

### 2.2.3 InfraEstrutura como serviço - IaaS

O modelo de infraestrutura como serviço (*Infrastructure as a Service* - IaaS) [20, 21] apresenta ao usuário do sistema computacional uma interface onde ele pode solicitar a alocação de recursos computacionais de acordo com sua demanda sem a necessidade de uma prévia contratação destes recursos. Como ilustrado na Figura 2.3, IaaS é a camada de abstração mais proxima do *kernel* do sistema e o *hardware* em comparação com os modelos SaaS e PaaS. O cliente de **IaaS** administra um parque de servidores virtuais que pode expandir ou contrair de acordo com a demanda, ou disponibilidade de recursos do provedor. Observe que um cliente de IaaS pode tanto ser um administrador como um serviço, ou sistema, computacional totalmente automatizado. Entre os os provedores deste tipo de serviço, estão a Amazon Web Services com seu serviço de **EC2** [28] e o **Enomaly** [29], além disso, existem sistemas de código aberto para a construção de arquiteturas de IaaS como o **Eucalyptus** [30], **Opennebula** [31] e **OpenStack** [32].

#### Eucalyptus

O **Eucalyptus** é um sistema de código aberto para a construção de arquiteturas de IaaS, sendo uma das primeiras plataformas abertas disponibilizadas e originalmente oferecia suporte apenas ao monitor de máquinas virtuais Xen, mas em sua versão **2.0** passou a oferecer suporte ao **KVM** [33]. A motivação para sua criação foi fornecer meios para os pesquisadores estudarem os problemas que afetam os arquiteturas de IaaS [30]. A arquitetura do **Eucalyptus** é composta por quatro componentes principais, são eles:

- Controle de Nó computacional: controla a execução, inspeção e término de uma máquina virtual em seu nó computacional;
- Controle do *Cluster*: executa no *cluster front-end* e recolhe informações dos Controles de Nós e escalona máquinas virtuais em um Controle do Nó específico, além de gerenciar a rede virtual do cluster;

- Controle de Armazenamento: Componente que executa *cluster front-end* e implementa o serviço de armazenamento *put/get*;
- (**Walrus**): um serviço de armazenamento *put/get* que implementa a interface do S3 [34] da Amazon;
- Controle da Cloud: ponto de entrada dos usuários e administradores na *Cloud* (arquiteturas de IaaS). Consulta o controle do cluster para pegar informações sobre recursos e toma decisões de escalonamento de alto nível implementando estas decisões através de requisições ao controle do *Cluster*

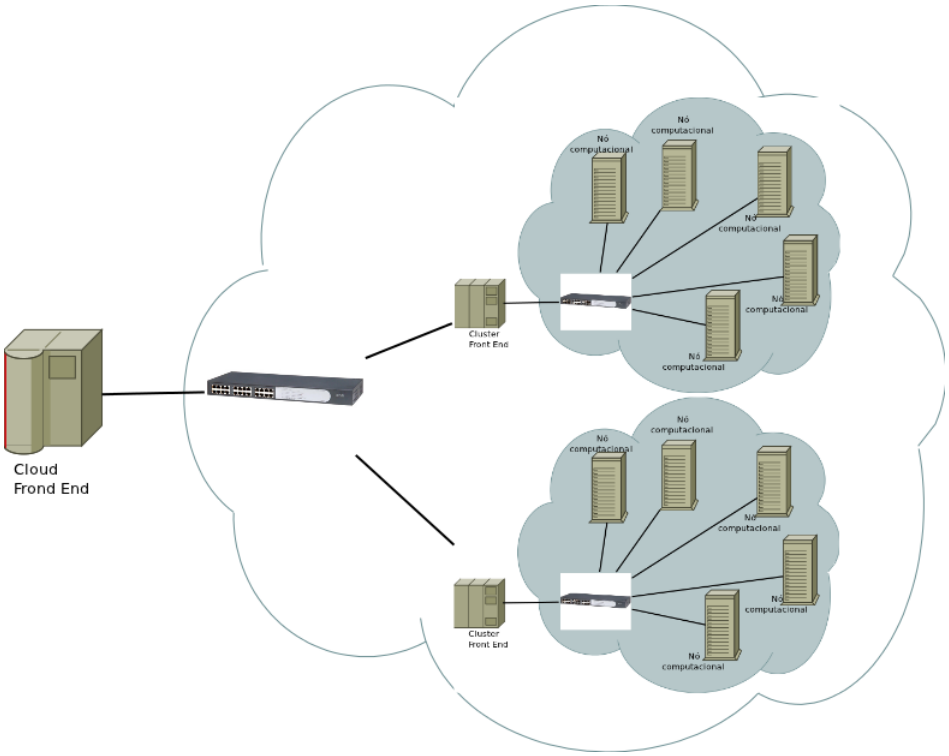


Figura 2.4: Arquitetura do arquitetura de **IaaS** usando **Eucalyptus** com dois *clusters*

A Figura 2.4 apresenta a arquitetura física de uma instalação do **Eucalyptus** com dois clusters, nela é possível observar o *Cloud Front-End* que executa o Controle da Cloud. O *Cloud Front-End* é interconectado com os *Clusters Front-Ends* através de um comutador. Onde ainda cada um *Cluster Front-End* executa o Controle do *Cluster*, sendo responsável por atender as requisições do *Cloud Front-End* e se comunicar com os Controle de Nó computacional que executam em cada um dos nós do cluster.

O **Walrus** é o componente de alto nível do sistema de armazenamento que executa no *Cloud Front-End*, visto na Figura 2.4, enquanto que cada um dos *Cluster Front-Ends* do sistema deve executar o Controle de Armazenamento.

**OpenNebula**

O **OpenNebula** é ums sistema para gerenciamento de infraestrutura virtualizada, que facilita a construção de um arquiteturas de IaaS privado ou híbrido. Ele é um sistema open source que possibilita que uma organização implemente o modelo de uso de IaaS em seus clusters, oferecendo deste modo flexibilidade e agilidade aos desenvolvedores e administradores de sistemas. O **OpenNebula** possui dois tipos de máquinas, o *front end*, responsável por receber as requisições dos clientes através de uma API XML-RPC e os nós computacionais que executam efetivamente as máquinas virtuais. Ele oferece ainda interface de controle compatível com a do **EC2** onde os usuários podem requisitar alocação e desalocação de recursos computacionais de acordo com sua demanda atual.

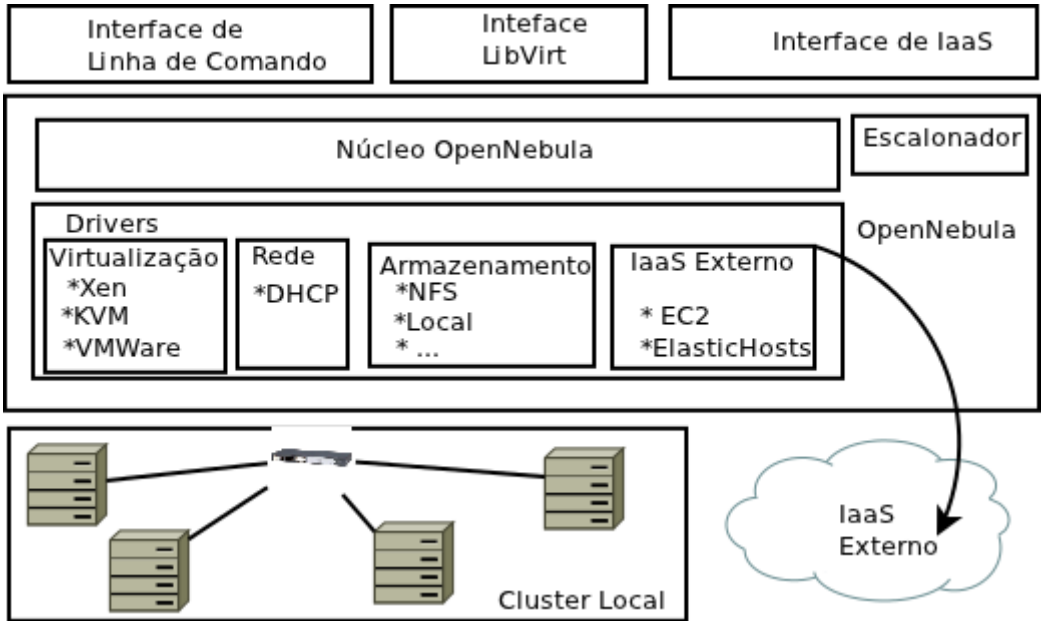


Figura 2.5: Arquitetura do arquitetura de IaaS **OpenNebula**

A Figura 2.5 apresenta uma visão simplificada da arquitetura do arquitetura do **OpenNebula**, nela é possível observar no nível mais baixo a infraestrutura local, dela faz parte também o sistema operacional, o monitor de máquina virtual e um possível provedor externo de IaaS, sobre estes encontra-se o **OpenNebula** propriamente dito, este ainda subdivido em *drivers*, escalonador e núcleo. Os *drivers* do **OpenNebula** servem como "cola", ou seja eles são responsáveis por efetivamente executarem as tarefas do sistema de gerenciamento sobre a infraestrutura física, ainda com relação aos *drivers* deve ser observado que em sua versão atual o **OpenNebula** oferece suporte a três monitores de máquinas virtuais, são eles:

- Xen
- KVM

- VMWare

O núcleo do **OpenNebula** é o componente da arquitetura responsável por receber as requisições das interfaces com o usuário e realizar as operações necessárias para servir essas requisições. Nos casos onde a requisição é uma alocação de máquina virtual ele deve primeiro contatar o Escalonador para descobrir em qual recursos físico a nova máquina virtual deve ser iniciada. Por fim, no topo da Figura 2.5, encontram-se as interfaces com o **OpenNebula**, são através destas que os usuários do sistema interagem com o mesmo.

A arquitetura modular do **OpenNebula** facilita a introdução de novos recursos como novos monitores de máquina virtuais ou a alteração do escalonador utilizado por este. Um exemplo deste tipo de alteração, é a substituição do escalonador padrão do sistema de gerenciamento pelo **Haizea** no projeto **RESERVOIR** [35, 31]. Sendo que assim como o **Eucalyptus**, o **OpenNebula** oferece suporte a múltiplos clusters [36].

### OpenStack

O **OpenStack** [32] é um sistema de código livre para o desenvolvimento de arquiteturas de computação na nuvem escaláveis. Ele pode ser utilizado para o projeto de arquiteturas de IaaS públicos ou privados e atualmente é dividido em dois projetos co-relacionados, são eles:

- *OpenStack Compute* [37]: Responsável por gerenciar as máquinas virtuais e rede;
- *OpenStack Object Storage* [38]: Serviço de armazenamento redundante e escalável.

O *OpenStack Compute*, também conhecido como **Nova**, é uma ferramenta para provisionar e gerenciar grandes redes de máquinas virtuais. Ela oferece o suporte necessário para o gerenciamento de uma arquitetura de IaaS, como por exemplo, a execução de máquinas virtuais e gerenciamento de redes virtuais. Ela tem como objetivo ser independente da tecnologia de virtualização de modo que define um conjunto de drivers que acessam a técnica de virtualização e exporta as funcionalidades através de sua API. Atualmente o **Nova** oferece suporte a sete monitores de máquinas virtuais distintos, são eles:

- Xen;
- KVM;
- LXC;
- Hyper-V 2008;
- QEMU;
- User Mode Linux;
- VMWare.

Ainda com relação ao projeto **Nova**, as imagens das máquinas virtuais podem ser gerenciadas de duas maneiras distintas. A primeira é através do serviço *OpenStack Image* [39], ou **Glance**. O **Glance** é responsável por rotear as requisições dos clientes para um dos seus serviços de armazenamento, que podem ser um sistema de arquivo, armazenamento no **S3** da *Amazon*, HTTP ou *OpenStack Object Storage*. A outra forma como as imagens podem ser armazenadas é utilizando diretamente o serviço de armazenamento *OpenStack Object Storage*.

O serviço de armazenamento, o *OpenStack Object Storage*, também chamado de **Swift**, é um aplicativo de código livre para construir um sistema de armazenamento de objetos escalável e redundante usando cluster de computadores. O **Swift** não é um sistema de arquivos ou sistema de armazenamento de tempo real, mas um sistema de armazenamento de longo prazo e escalável. O sistema fica responsável pela replicação e consistência das escritas realizadas. Em termos seu uso o **Swift** oferece uma interface similar ao **S3**, ou seja os usuários podem escrever pares de chave/valor (limite de 4 KB) ou buscar um dado valor (objeto) fornecendo sua chave como parâmetro.

## 2.3 Serviços Web escaláveis

A criação e manutenção de serviços Web escaláveis representam ainda um grande desafio para projetistas, desenvolvedores e administradores destes serviços, pois não se pode prever o crescimento da demanda por poder computacional relacionado ao aumento do número de usuários do serviço [20]. Somando-se a isso existe a dificuldade inerente de se projetar e construir sistemas escaláveis com o número de usuários e os custos associados ao provisionalmente errado da capacidade computacional para o novo serviço serviço.

Atualmente, os recursos computacionais utilizados na construção de serviços escaláveis são organizados como clusters de computadores, tendo esta maneira de organizar os recursos de *hardware* sido utilizada com sucesso tanto na academia quanto na indústria como forma de oferecer serviços que necessitem de mais poder computacional do que pode ser encontrado em um servidor comum. Contudo, apesar de sua larga adoção, a programação de sistemas eficientes nesta arquitetura computacional está longe de ser uma tarefa trivial.

Neste cenário, o único padrão de fato é a construção serviço como um sistema distribuído usando passagem de mensagem para realizar a troca de dados entre os diversos componentes sistemas. Por exemplo, na computação científica essa troca de mensagens é feita através do padrão **MPI**. Existem ainda trabalhos que buscam tornar mais simples o desenvolvimento de sistemas para essa classe de computadores em troca de um aumento no *overhead* mais nenhum ainda desponta como uma solução de propósito geral.

### 2.3.1 Arquitetura Lógica de um Sistema de Transmissão de vídeo sob demanda

Um sistema de transmissão de vídeo sob demanda (VsD) é um serviço Web de tempo real não-crítico (*soft real-time*), modelado conceitualmente como uma aplicação cliente servidor, onde o cliente solicita ao servidor que esse lhe envio

o arquivo de vídeo desejado. Os requisitos de tempo real deste tipo de serviço surgem da necessidade de o servidor transmitir os dados enquanto concorrentemente o cliente está exibindo o vídeo.

Os requisitos de tempo real e QoS, somados as questões de escalabilidade, torna necessário construir este tipo de serviço Web de maneira mais específica. Uma possível maneira de se projetar este serviço de modo a atender estes requisitos é através da introdução, por exemplo, servidores de cache entre o servidor de vídeo e os clientes.

### GloVE: Um sistema de Tempo-Real Não-crítico para Transmissão de vídeo sob demanda na Internet

O GloVE (*Global Video Environment*) [40, 41] é um sistema de transmissão de vídeo sob demanda, baseado na técnica de Cache de Vídeo Cooperativa Colapsada (CVC-C) [42]. Nele os servidores de cache tradicionais são substituídos por Distribuidores, que passam a ser os componentes responsáveis por enviar o vídeo aos clientes, uma das vantagens desses distribuidores é que novos podem ser introduzidos no sistema enquanto esta continua funcionado com o intuito de aumentar a capacidade de transmissão do serviço. O **GloVE** é composto por três componentes principais:

- Servidor;
- Distribuidor (Cache);
- Cliente.

A Figura 2.6 apresenta uma das possíveis organizações destes componentes em uma rede de computadores. Neste figura temos o servidor central que armazena os vídeos em disco, onde os números próximos as setas indicam que dois fluxos distintos estão sendo transmitidos para cada um dos clusters. Os Distribuidores são organizados em dois clusters, cada um contendo quatro nós computacionais, é possível observar pela figura que apenas os distribuidores recebem fluxos de vídeo do Servidor. O sistema **GloVE** implementa o reuso do fluxo oriundo do servidor, como pode ser visto observando-se os números próximos as setas que ligam os distribuidores aos clientes. No cenário exemplificado pela Figura 2.6 os clientes somente podem receber vídeos através dos nós distribuidores, observe porém que esta simplificação tem como objetivo apenas tornar mais clara a explicação e não representa uma limitação imposta na arquitetura do sistema de distribuição. Por fim é importante ressaltar que é o servidor que informa ao cliente a qual distribuídor ele deve solicitar o vídeo.

### SMART: Um sistema de Tempo-Real Não-crítico para Transmissão de vídeo sob demanda na Internet usando virtualização

O **SMART** [1] sistema para transmissão de vídeo sob demanda escalável, projetado em torno de técnicas de virtualização. Nele, os nós do cluster executam um ou mais contêineres, como pode se observado na Figura 2.7, cada contêiner executando uma instância da aplicação de transmissão de vídeo e assumindo ainda que, cada instância do aplicação de transmissão ficaria responsável por apenas um único vídeo.

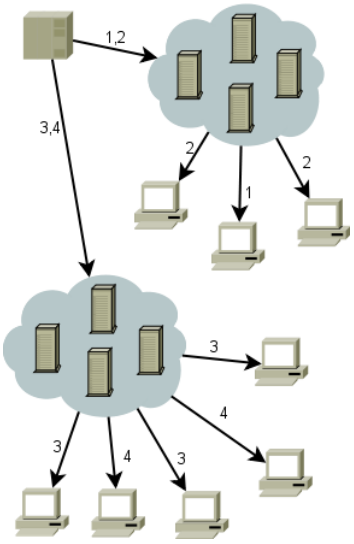


Figura 2.6: Organização multi-cluster do sistema de distribuição **GloVE**

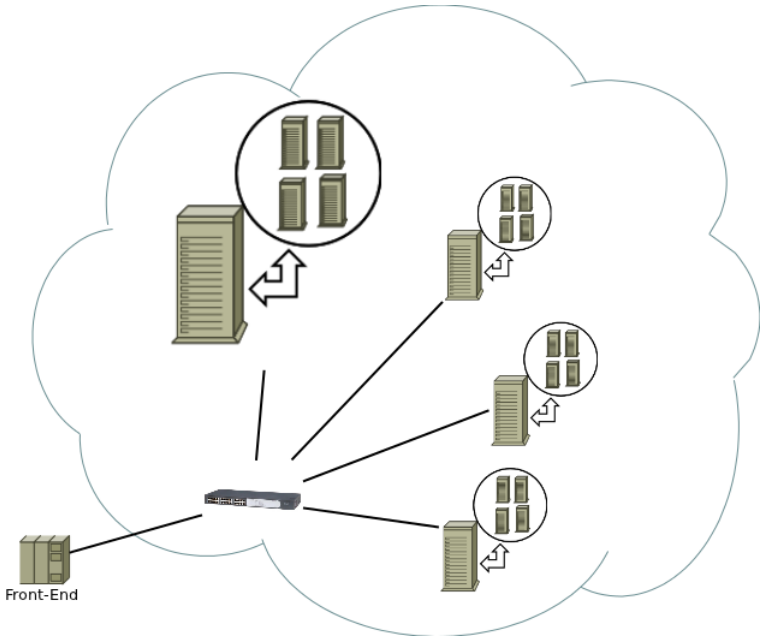


Figura 2.7: Cluster de servidores virtualizados utilizados no sistema de transmissão de vídeo sob demanda **SMART**

O uso de virtualização permitiu ao sistema **SMART** fornecer as garantias de QoS necessárias para a construção de um serviço de transmissão de vídeo sob demanda. Com relação a escalabilidade, o projeto do **SMART** o torna escalável uma vez que para aumentar a capacidade de transmissão do sistema

basta executar um novo contêiner, dado é claro que existam recursos computacionais disponíveis no cluster para que o contêiner seja executado.

# Capítulo 3

# Revisão Bibliográfica e Trabalhos Relacionados

## 3.1 Avaliação de desempenho de servidores virtualizados

Esta seção apresenta importantes trabalhos onde foram realizadas avaliações sistemáticas do desempenho de diferentes monitores de máquinas virtuais. Estes trabalhos são relevantes dentro do escopo deste relatório tendo em vista que máquinas virtuais representam o bloco básico na alocação de recursos em uma arquitetura de IaaS. Sendo necessário uma compreensão de como os aplicativos utilizados na avaliação de um determinadas arquiteturas de IaaS se comportam quando executados isoladamente com o modelo, ou modelos de virtualização suportados pela arquitetura computacional que se deseja estudar.

Casazza et al. [43] apresentam uma metodologia para a construção de *benchmarks* para arquiteturas virtualizadas, o **vConsolidate**, onde um servidor físico consolida múltiplas máquinas virtuais. A carga de trabalho do **vConsolidate** é composta por uma máquina virtual executando um servidor Web, outra com um servidor de e-mail e uma terceira executando um banco de dados. Os autores defendem que os *benchmarks* sejam otimizados em uma arquitetura não virtualizada antes de eles serem utilizados e que os resultados obtidos desta execução devam ser utilizados como base para a avaliação consolidada. Uma vez que se tem a avaliação inicial do desempenho os autores definiram que o resultado final da execução dos *benchmarks* será uma média ponderada dos resultados obtidos com a execução dentro da arquitetura virtualizado, observando porém que os pesos são fixos e determinados pela carga de trabalho de cada um dos *benchmarks*. Embora não proposto como um *benchmark* para a indústria, o **vConsolidate** apresenta uma metodologia para se construir um *benchmark* para uma arquitetura virtualizado compartilhado.

Padilla et al. [44] realizaram uma avaliação comparativa entre **Xen** e **OpenVZ**, usando o *benchmark* **RUBiS**. Em paralelo com a execução do *benchmark*, obtiveram um perfil do sistema utilizando o **Oprofile**. Os resultados apresentados

no trabalho mostraram que apesar do *benchmark* utilizando o **Xen** ter obtido uma vazão similar a do nó físico e a também do sistema usando **OpenVZ**, o mesmo não ocorre com a latência (tempo de resposta). Esta apresentou apenas um pequeno aumento tanto para o nó físico como para o **OpenVZ**, enquanto que em alguns cenários, o **Xen** apresentou um aumento de mais de 600% na latência e o consumo de CPU foi aproximadamente o dobro. A avaliação dos experimentos utilizando o **Oprofile** por sua vez mostrou uma diferença significativa no número falhas de **cache L2** para os DomU do **Xen** em comparação ao nó físico, ao **OpenVZ** e ao Dom0 do **Xen** para o *kernel* do **Linux**. Devido a esta diferença significativa os autores acreditam que a sobrecarga do **Xen** seja causada pelo grande número de *falhas* de **cache L2**, enquanto para o **OpenVZ** a sobrecarga é baixa e chega a ser desprezível em muitos dos cenários estudados.

Apparao et al. [45] realizaram um estudo experimental do *benchmark* **vConsolidate** [43]. A carga de trabalho do **vConsolidate** utilizada no estudo foram máquinas virtuais contendo um servidor Web, um servidor de e-mail, um servidor de banco de dados, um servidor Java (SPECjbb) e uma máquina virtual sem carga (*idle*). O *benchmark* foi executado sobre um servidor duo processado baseado no Core 2 Duo, com suporte à virtualização habilitado e o monitor de máquinas virtuais utilizado foi o **Xen**. Foram levantados resultados relativos a degradação de desempenho tanto em termos de vazão das requisições atendidas como do uso do processador devido à consolidação dos servidores virtuais. Em relação a queda de vazão nos cenários consolidados os autores apresentam como possíveis causas a contenção no acesso aos recursos da plataforma (núcleo de processamento, cache, memória e rede) ou a sobrecarga intrínseca a virtualização. Por fim, os autores apresentaram uma proposta inicial de um modelo para avaliação de desempenho de um servidor consolidado, com o objetivo de prever o desempenho de cargas de trabalho virtualizadas (*virtualized workloads*) em futuras plataformas.

## 3.2 Avaliação de desempenho de cluster de servidores virtualizados

Esta seção apresenta importantes trabalhos em que foram realizadas avaliações sistemáticas de arquiteturas de clusters de computadores, onde os nós computacionais eram servidores com suporte para virtualização. Estes trabalhos são relevantes dentro do escopo deste relatório tendo em vista, tanto as questões relativas a semelhança desta arquitetura física e a arquitetura que suporta os serviços de IaaS, como também auxilia no entendimento de como o uso de virtualização impacta no desempenho das aplicações que executam sobre um cluster.

Youseff et al. [46], investigaram experimentalmente as sobrecargas causadas pelo MMV **Xen** no desempenho de *kernels* e aplicativos de HPC, sendo que no referido estudo cada nó do naquele *cluster* executou apenas uma única máquina virtual. Os resultados apresentados pelos autores demonstraram que apesar da versão do **Xen** utilizada nos experimentos ser eficiente em termos de injeção de pacotes na rede (trafego de rede unidirecional), já que se aproxima de um núcleo do Linux otimizado para arquiteturas de **HPC** usando **MPI**, o mesmo não acontece com o trafego bidirecional, o que demonstra uma maior sobrecarga na recepção de pacotes por parte do **Xen**. Os experimentos utilizando o apli-

cativo real foram feitos com o **LU** da suíte **Linpack**, sendo que os resultados apresentados demonstraram que o **Xen** no cenário estudado tem desempenho próximo ao do nó físico, porém ainda assim inferior.

Nussbaum et al. [47] realizaram um estudo comparativo entre **KVM** e **Xen**, para verificar a viabilidade destes em um ambiente de HPC e destacar áreas onde esses monitores necessitam de melhorias. Com relação as vantagens que o uso de virtualização trariam ao ambiente de HPC, elas seriam a flexibilidade na configuração das máquinas e o suporte para *checkpoint*/migração da máquina virtual. Em sua avaliação experimental, os autores inicialmente avaliaram, através de *micro benchmarks* o uso do processador, o acesso a disco e a vazão de rede; posteriormente, foram executados benchmarks de HPC como o **Linpack**. Os experimentos tanto do *micro benchmark* de rede como os dos benchmarks de HPC, demonstraram que o KVM consegue uma vazão de rede superior ao **Xen**, seja este último com suporte para virtualização em *hardware* ou paravirtualizado.

### 3.2.1 Avaliação de desempenho para arquiteturas de Infraestrutura como Serviço para aplicações Web

Devido as suas características e crescente demandas por poder computacional os serviços Web tem sido de grande utilizade em diversas avaliações publicadas sobre arquiteturas de IaaS. A seguir, são apresentados alguns trabalhos que buscaram experimentalmente avaliar o impacto desta nova arquitetura sobre esses serviços.

Kossmann et al. [48], realizaram um estudo de diferentes arquiteturas de computação na nuvem para aplicativos de banco de dados, usando o benchmark **TPC-W**. Como o objetivo dos autores era avaliar o desempenho de ambientes de computação na nuvem para aplicativos de processamento transacional, eles avaliaram tanto soluções de IaaS como as de PaaS. Os cenários utilizados nos experimentos variaram de soluções puras de IaaS como o **EC2**, o uso do EC2 com outros serviços de armazenamento (**S3**) ou banco de dados relacional na nuvem (**RDS**), a ambientes de PaaS como o **Azure** da Microsoft e o **AppEng** da Google. Para realizar os experimentos os autores reimplementaram o **TPC-W**, para que este pudesse utilizar as características de cada um dos ambientes, sendo que para o **Azure** foi necessário reescrever o mesmo em outra linguagem. O *benchmark* escolhido reporta informações com relação a vazão total de consultas válidas por segundo e custo de consulta por cliente (desempenho). Para os cenários do **EC2** e do **Azure**, as máquinas virtuais tinham aproximadamente a mesma capacidade computacional. O artigo reporta resultados para custo e escalabilidade, este último como volume total de requisições atendidas com sucesso, sendo que os únicos cenários que demonstraram escabilidade foram o do **EC2** usando sistema de armazenamento **S3** e o do Azure. Ainda, mostraram como em um mesmo ambiente, no caso o **EC2** da *Amazon*, diferenças nos serviços utilizados afetam o desempenho do aplicativo.

Sobel et al. [49], desenvolveram o **Cloudstone**, um *benchmark* para arquiteturas de IaaS. O **Cloudstone** é composto por um aplicativo Web 2.0, um conjunto de ferramentas para automatizar a geração de carga e medições de desempenho em diferentes cenários. O aplicativo Web 2.0 é o Olio e para aquele trabalho os autores realizaram duas implementações do Olio, uma em *Ruby on Rails* e outra em PHP. Um ponto importante do referido trabalho é a reco-

mendação por parte dos autores de se utilizar métricas baseadas em custo em dálares como parâmetro de avaliação. O artigo apresenta resultados de ambas as versões do **Cloudstone** para diferentes configurações no EC2; foram executados seis cenários para Ruby e um para PHP, e com base neles é possível observar que os cenários com Ruby obtiveram um desempenho pior em média que os com PHP e que o desempenho do Ruby no EC2 é melhor com máquinas de maior capacidade de processamento.

Ueda e Nakatani [50], realizaram uma comparação experimental entre as plataformas para construção de arquiteturas de IaaS Eucalyptus e OpenNebula, ambas usando o Xen como monitor de máquina virtual, realizando ainda uma comparação destes resultados com os obtidos no EC2, onde ainda para o OpenNebula foram feitas avaliações alterando a forma como as imagens das máquinas virtuais eram armazenadas. Eles construíram um *benchmark* que modela o serviço web Wikipédia utilizando ferramentas de código livre, além disso, os autores realizaram uma avaliação do tempo médio de provisonamento de uma instância virtual nas três arquiteturas. Os resultados obtidos pelos autores mostraram para as soluções de código livre que foram avaliadas a fundo como diferenças na construção da arquitetura impactaram tanto o desempenho da aplicação quando o sistema esta em funcionamento, como quando se necessita provisionar novas máquinas virtuais.

# Capítulo 4

# Serviço de Transmissão de vídeo sob demanda

Neste capítulo é descrito os componentes do serviço de distribuição de vídeo sob demanda escalavél, detalhando como eles interagem entre si e a arquitetura virtualizada.

## 4.1 Serviço de Transmissão de vídeo sob demanda Escalável em um ambiente virtualizado

A implementação de um serviço de transmissão de vídeo sob demanda organizado como apresentando na seção 2.3.1, sobre uma arquitetura virtualizada de forma a atender a demanda dos usuários sem com isso desperdiçar recursos computacionais mantendo máquinas ociosas ativas é feita através da execução dentro de máquinas virtuais do componente de distribuição de vídeo. Por exemplo, os servidores virtuais neste cenário assumiriam, por exemplo, a função dos servidores cache e passariam então a responder pelo envio dos fluxos de vídeos aos clientes, desta maneira o serviço de transmissão será composto por três componentes principais, são eles:

- Servidor de gerenciamento;
- Servidor de distribuição;
- Cliente

O **servidor de gerenciamento** é o componente responsável pelo gerenciamento das máquinas virtuais e definição de qual vídeo ela deve exibir [1], ele recebe requisições de vídeo dos clientes e baseado nestas requisições deve balancear a carga no sistema. Ele deve determinar se existe um **servidor de distribuição** que possa atender este **cliente** e caso exista, informar ao cliente qual será este servidor, e no caso onde não for possível utilizar um servidor de

[1] Pode se definido isso através da imagem da máquina virtual ou pela aplicação em tempo de execução

distribuição ativo, o servidor de gerenciamento deverá alocar uma nova máquina virtual. O **servidor de distribuição** é o componente do sistema de transmissão que executa obrigatoriamente dentro da máquina virtual, ele é responsável pela transmissão do vídeo para os clientes e por informar ao servidor de gerenciamento quantos clientes ele tem ativos.

### 4.1.1 Métricas de Avaliação

As métricas de avaliação do serviço de transmissão de vídeo sob demanda escalável, também tanto das características de implementação de uma arquitetura virtualizada para a aplicação de transmissão de VsD as métricas compreendem:

- Percentual de clientes atendidos (**PcA**): Percentual de clientes que receberam vídeo;
- Percentual de Banda agregada total utilizada (**AggBand**): Fração utilizada da banda disponível no sistema.

## 4.2 Considerações Finais

Esta seção detalhou o trabalho proposto para a tese. Descrevendo um serviço Web de tempo real (transmissão de vídeo sob demanda) na Seção 4.1 a serem utilizados como mecanismos para aferir quantitativamente o desempenho das diferentes arquiteturas de IaaS. No próximo Capítulo, são apresentados os resultados experimentais preliminares que foram realizados dentro do escopo do serviço de transmissão de vídeo sob demanda elástico descrito na Seção 4.1.

# Capítulo 5

# Avaliação Experimental Preliminar

Neste capítulo são apresentados alguns resultados dos experimentos preliminares que foram realizados dentro do LCP ao longo dos últimos dois anos. Os resultados experimentais apresentados são focados na construção do serviço de transmissão de vídeo sob demanda, discutido na Seção 4.1. Esses resultados servem como base no projeto e avaliação do sistema de transmissão.

## 5.1 Ambiente Experimental

O ambiente experimental utilizado na validação foi composto por quatro nós computacionais, interconectados por um *switch Gigabit Ethernet*, cujas características dos nós encontram-se descritas na Tabela 5.1 (com o tipo do componente na primeira coluna e o modelo utilizado na segunda). Os nós executam o sistema operacional e mecanismos de virtualização descritos a seguir:

- Sistema Operacional: CentOS 5.4;;
- Versão do *kernel* do Linux : 2.6.18;
- Versão do *patch* OpenVZ: 028*stable*057.2;
- Versão do Xen: 3.4.1;
- Versão do KMV: 2.6.18.

A aplicação utilizada nos experimentos foi o servidor **HTTP NGINX** com suporte para transmissão de streaming de vídeos codificados em **H264** e utilizado a ferramenta de *profiling* **OProfile** para se contabilizar os tipos de eventos (chamadas). Os experimentos foram feitos com vídeos de três taxas distintas, são elas:

- 2.8 Mbps;
- 1 Mbps;
- 512 Kbps.

Tabela 5.1: Nó computacional Dual Xeon Quad (Arquitetura Core2)

| Componente | Modelo |
|---|---|
| Processador | Intel Pentium Xeon $E5410$ 2.33 GHz |
| Duração Teórica do Ciclo | $0,43$ ns |
| Cache L2 | 12 MB |
| FSB | 1333 MHz |
| Memória DRAM | $4 * 2$ GB DDR2 ECC |
| Interface de Rede | $2 * 10/100/1000$ Mbps |
| Unidade de Disco | 500 GB SATA 5400 RPM |

## 5.2 Avaliação das técnicas de virtualização utilizando um servidor de streaming de vídeos

Nesta seção estão sumarizados os resultados obtidos durante os experimentos. Inicialmente são apresentados os resultados para o vídeo com taxa de **2.8 Mbps** e quando se fizer necessário, resultados de experimentos realizados utilizando vídeo com outras taxas. Em todos os experimentos foram realizadas **10** execuções de cada um dos experimentos e com intervalo de coleta do perfil da transmissão de **60** segundos.

### 5.2.1 Transmissão sem virtualização

A execução do servidor de vídeo sem uso de técnicas de virtualização afere a capacidade efetiva, neste caso, definida como o total de clientes atendidos simultaneamente. Sendo observardado que a capacidade de transmissão do Linux sobre o nó físico e do Dom0 do Xen eram as mesmas.

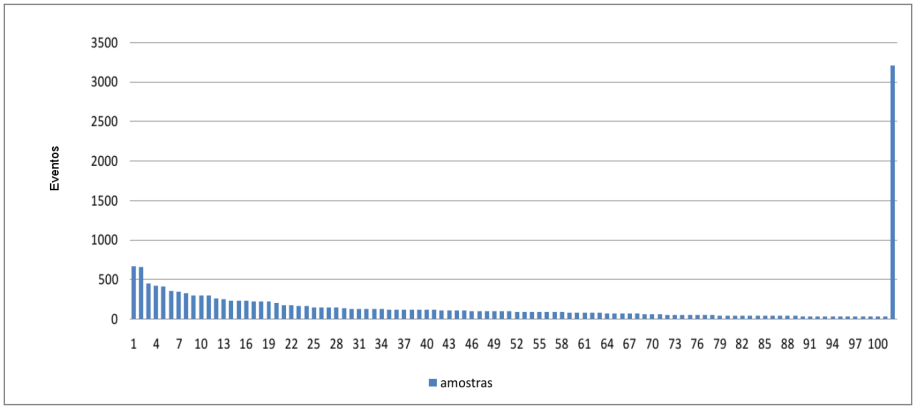


Figura 5.1: Distribuição de eventos para 320 clientes no servidor de vídeo sem virtualização

O servidor atendeu um total de **320 clientes** assistindo um vídeo com taxa de **2.8 Mbps**. Onde os clientes entraram no sistema sucessivamente em um curto intervalo de tempo. Deste modo a banda passante utilizada do servidor, após a entrada do ultimo cliente, durante os experimentos foi de 896 Mbps.

A Figura 5.1 [1] apresenta a distribuição de eventos capturados pela ferramenta de *profiling* durante uma execução do sistema de distribuição de vídeos. Neste cenário foram capturados para o perfil **16990** eventos divididos em **533** classes distintas durante os 60 segundos considerados, sendo apresentado na figura apenas 99 destas classes e os demais eventos agrupados no **label 100**.

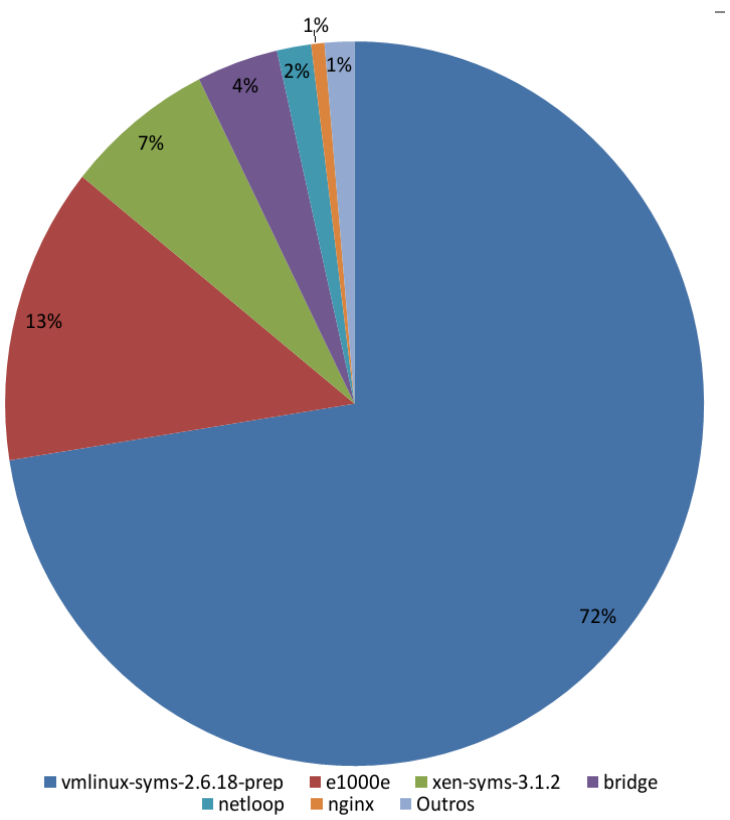


Figura 5.2: Amostragem de eventos para 320 clientes no servidor de vídeo sem virtualização

Os eventos de maior ocorrência na series de dados apresentada pela Figura 5.1 são os causados pelo envio de mensagens do *kernel* do Linux para a interface de rede como, por exemplo, eventos ligados a interrupção da interface de rede como o *e1000_irq_enable*, representando na Figura 5.1 como o evento de **classe 1**.

Devido à quantidade de classes e eventos capturados, a Figura 5.1 apenas fornece, a distribuição quantitativa dos mesmos. Na Figura 5.2, verifica-se os sete programas que mais geraram símbolos no perfil. E como foi utilizado o Dom0 é possível ver que este gerou 7 % dos eventos vistos na Figura 5.1.

Na Figura 5.2, pode-se observar que os eventos ocorridos com maior frequência no intervalo de tempo considerado foram os eventos ligados ao envio de pacotes pela interface de rede, sejam os vinculados à interface propriamente dita

[1] Uso números no rotulo para facilitar a leitura da figura

e classificados como e1000e que representaram 13% do total ou os executados pelo sistema operacional como parte do processamento da pilha de protocolos com 72%. Por fim, é importante salientar que a aplicação de transmissão de *streaming* de vídeo (servidor web NGINX) gerou apenas 1% do total de eventos no intervalo de interesse. Isso demonstra que a maior sobrecarga de processamento é dada pelo processamento ligado ao dispositivo de entrada/saída, no caso, a interface de rede. É importante manter em mente esses resultados pois os mesmos facilitarão a comparação com as outras técnicas de virtualização.

Foi obtido também para esta configuração, o perfil de uso dos núcleos de processamento disponíveis, que foi capturado durante os experimentos descritos anteriormente, desta forma são apresentados os perfil de uso das unidades de processamento para os casos onde existem 100, 200 e 300 clientes recebendo vídeos do servidor (neste caso o número de clientes é igual ao número de fluxos mantidos pelo servidor). Por fim, é importante observar que o **NGINX** é um servidor **HTTP** baseado em eventos e deste modo não possui múltiplas threads ou processos para realizar o atendimento das requisições **HTTP** como no caso do Apache, isso implica que o desempenho do servidor **HTTP** é diretamente ligada ao desempenho do núcleo de processamento e o uso da hierarquia de memória.

A Figura 5.3 apresenta a carga média de processamento no sistema e a carga de processamento no núcleo em que o **NGINX** executa, para o intervalo de tempo entre 50 segundos e 150 segundos de um perfil total de 300 segundos capturado durante os experimentos, sendo que os clientes começavam a entrar a requisitar vídeo depois de 10 segundos e quando chegamos a 50 segundos todos o clientes já solicitaram o envio do vídeo. O pico observado na figura deve-se às requisições HTTP dos últimos clientes que chegaram, que começam a receber pedaços de vídeo em uma taxa superior a de exibição do vídeo. A Figura 5.3 também mostra que o servidor **NGINX** consegue utilizar apenas um dois 8 núcleos de processamento disponíveis.

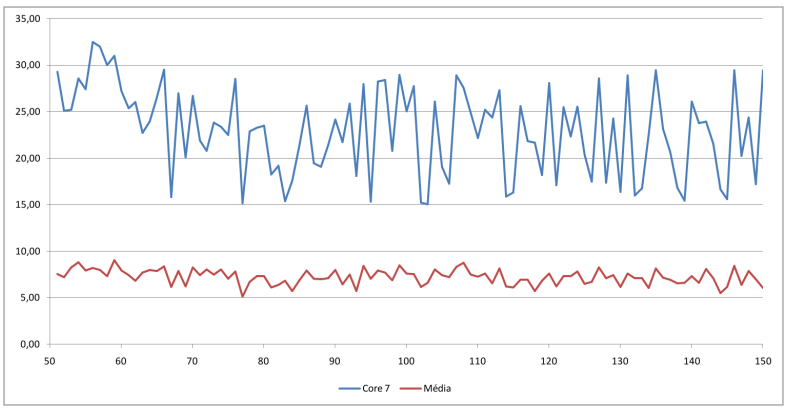


Figura 5.3: Amostragem da carga de processamento média e carga de processamento no núcleo em que executa o NGINX para 100 clientes no servidor de vídeo sem virtualização

A Figura 5.4 apresenta a carga média de processamento no sistema e a

carga de processamento no núcleo em que o **NGINX** executa, para o intervalo de tempo entre 50 segundos e 150 segundos de um perfil total de 300 segundos capturado durante os experimentos, sendo que os clientes começavam a entrar a requisitar vídeo depois de 10 segundos e quando chegamos a 90 segundos todos o clientes ja solicitaram o envio do vídeo.

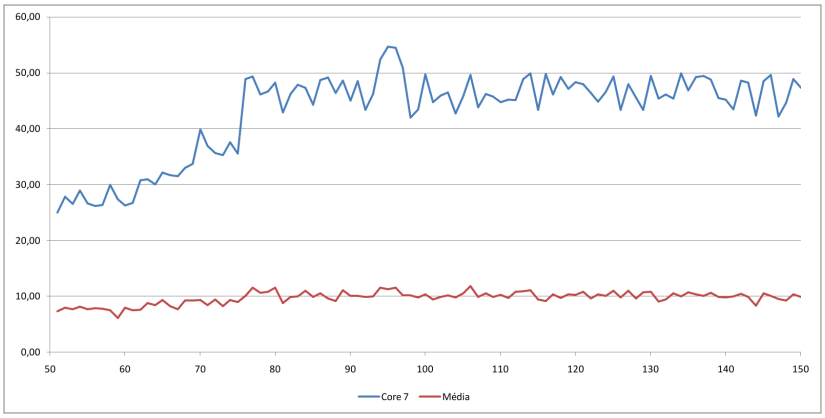


Figura 5.4: Amostragem da carga de processamento média e carga de processamento no núcleo em que executa o NGINX para 200 clientes no servidor de vídeo sem virtualização

A Figura 5.5 apresenta a carga média de processamento no sistema e a carga de processamento no núcleo em que o **NGINX** executa, para o intervalo de tempo entre 50 segundos e 150 segundos de um perfil total de 300 segundos capturado durante os experimentos, sendo que os clientes começavam a entrar a requisitar vídeo depois de 10 segundos e quando chegamos a 120 segundos todos o clientes ja solicitaram o envio do vídeo.

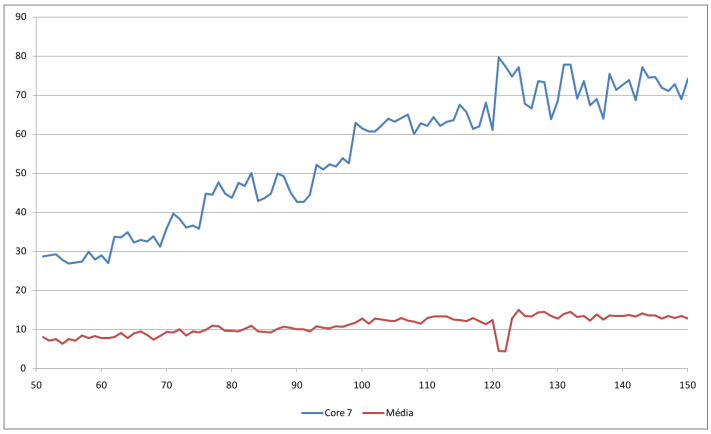


Figura 5.5: Amostragem da carga de processamento média e carga de processamento no núcleo em que executa o NGINX para 300 clientes no servidor de vídeo sem virtualização

Por fim, uma análise das figuras 5.3, 5.4 e 5.5, permite inferir que o custo computacional para realizar o atendimento das requisições **HTTP** para envio de vídeo é proporcional ao número de clientes que encontram-se recebendo vídeo do servidor no cenário experimental utilizado. Observe ainda que, apesar do servidor suportar enviar o vídeo para 320 clientes, para a análise de custo computacional, apenas foram apresentados resultados para 100, 200 e 300 clientes.

### 5.2.2 Transmissão de vídeo usando a virtualização do tipo contêiner com o OpenVZ

O experimentos realizados utilizando o **OpenVZ** como mecanismo de virtualização e executando o servidor **HTTP NGINX** dentro de um contêiner obtiveram resultados similares a execução do servidor diretamente sobre o Linux não virtualizado, não apresentando sobrecarga considerável, e conseguindo alcançar uma vazão efetiva de 320 clientes também para um vídeo com taxa de **2.8 Mbps**. Com base nestes resultados são apresentados para o servidor de vídeo executando dentro de um contêiner OpenVZ apenas os totais de clientes atendidos. A Tabela 5.2 apresenta na primeira coluna o número de contêiner OpenVZ que foram criados, sendo na segunda coluna apresentando o total de clientes que foram servidos por cada contêineres e a terceira coluna apresenta apenas o total de clientes atendidos pelo nó físico, ou seja, Total de Clientes = Número de VM's * Total de Clientes por VM.

Tabela 5.2: Sumário do experimento para o OpenVZ com vídeo de taxa de 2.8 Mbps

| Número de VM's | Total de Clientes por VM | Total de Clientes no *hardware* |
|---|---|---|
| 1 | 320 | 320 |
| 2 | 160 | 320 |
| 4 | 80 | 320 |
| 8 | 40 | 320 |
| 320 | 1 | 320 |

Como pode ser observado na Tabela 5.2 o uso da tecnologia de contêiner, implementado pelo **OpenVZ**, não apresentou sobrecarga significativa na transmissão dos fluxos de vídeo, permitindo ao servidor enviar vídeo para 320 clientes simultaneamente. De modo que, com base neste resultado, e o conhecimento prévio dos autores e a descrições da implementação do **OpenVZ** disponíveis na literatura, não foi necessário realizar uma avaliação de perfil para essas execuções, pois os resultados experimentais obtidos demonstram que a indireção introduzida nos descritores de processos e escalonamento das tarefas em dois níveis não impactam o desempenho da aplicação de transmissão de straming de vídeo.

### 5.2.3 Transmissão de vídeo usando a virtualização do tipo paravirtualizada com o Xen

Os experimentos com monitor de máquinas virtuais **Xen**, possui dois objetivos distintos, em primeiro lugar deseja-se avaliar comparativamente para um

sistema de transmissão de vídeo os efeitos que a implementação da técnica de paravirtualização do **Xen** tem sobre um sistema de transmissão de vídeo e em segundo lugar verificar como se comporta essa mesma implementação quando variamos o número de máquinas virtuais concorrentemente ativas no *hardware*.

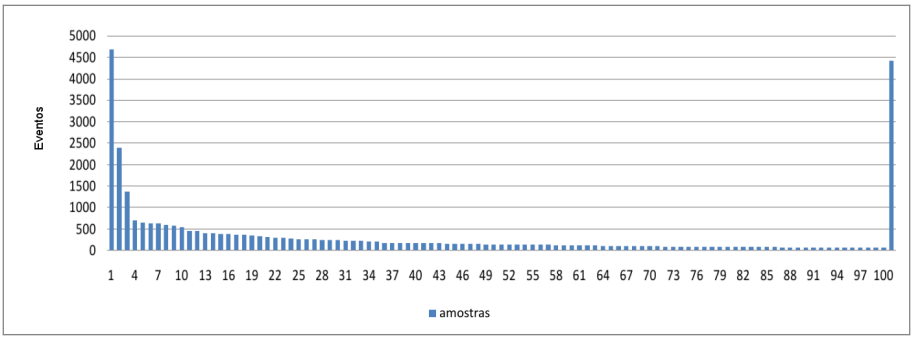


Figura 5.6: Distribuição de eventos para 242 clientes nó servidor de vídeo usando o monitor de máquinas virtuais Xen e uma única máquina virtual

Como comentado anteriormente na Seção 2.1.3, no **Xen** os domínios não privilegiados não possuem acesso direto aos dispositivos do sistema computacional, sendo necessário passarem pelo domínio privilegiado. Isto pode ser observado nos experimentos pelo fato de os eventos que mais ocorreram no sistema durante os experimentos terem sido o de acesso a *grant table* do **Xen** marcados na Figura 5.6 como os eventos de classe 1 ou *do_grant_table_op*. Sendo que a Figura 5.6 representa o experimento com apenas uma máquina virtual atendendo a todos os 242 clientes. Observe ainda, que como no cenário com o **OpenVZ**, devido a grande quantidade de classes e eventos capturados durante o intervalo de interesse dos experimentos, a Figura 5.6 fornecer a distribuição quantitativa dos mesmos.

Essa forma de controle ao acesso dos dispositivos imposta pela implementação de **Xen** faz com que a quantidade máxima de clientes suportados por uma única máquina virtual no *hardware* utilizado durante os experimentos nunca fosse superior a 242 clientes. A Tabela 5.3 apresenta os resultados obtidos para diferentes configurações no número de máquinas virtuais para o **Xen**, sendo na primeira coluna o número máquinas virtuais executadas concorrentemente, a segunda coluna contem o total de clientes servidos por cada máquina virtual nesta configuração e a terceira coluna apresenta apenas o total de clientes atendidos pelo no físico, ou seja, Total de Clientes = Número de VM's * Total de Clientes por VM. Ainda sobre a Tabela 5.3 os casos em que são apresentados número de clientes fracionários significa que duas das máquinas virtuais receberam um cliente a mais que as demais.

Com relação à quantidade de máquinas virtuais que foram executadas concorrentemente sobre o hardware durante os experimentos sumarizados na Tabela 5.3 verificou-se que ao menos para o serviço de transmissão de vídeos, verificou-se que com 12 máquinas virtuais executando concorrentemente a capacidade do **Xen** de transmitir dados os vídeos é comprometida e ele passa a atender menos que 242 clientes, sendo esse total de clientes dado pela capacidade de atendimento concorrente dos clientes na máquina virtual.

Tabela 5.3: Sumário do experimento para o Xen com vídeo de taxa de 2.8 Mbps

| Número de VM's | Total de Clientes por VM | Total de Clientes no *hardware* |
|---|---|---|
| 1 | 242 | 242 |
| 2 | 121 | 242 |
| 4 | 60, 5 | 242 |
| 8 | 30, 25 | 242 |

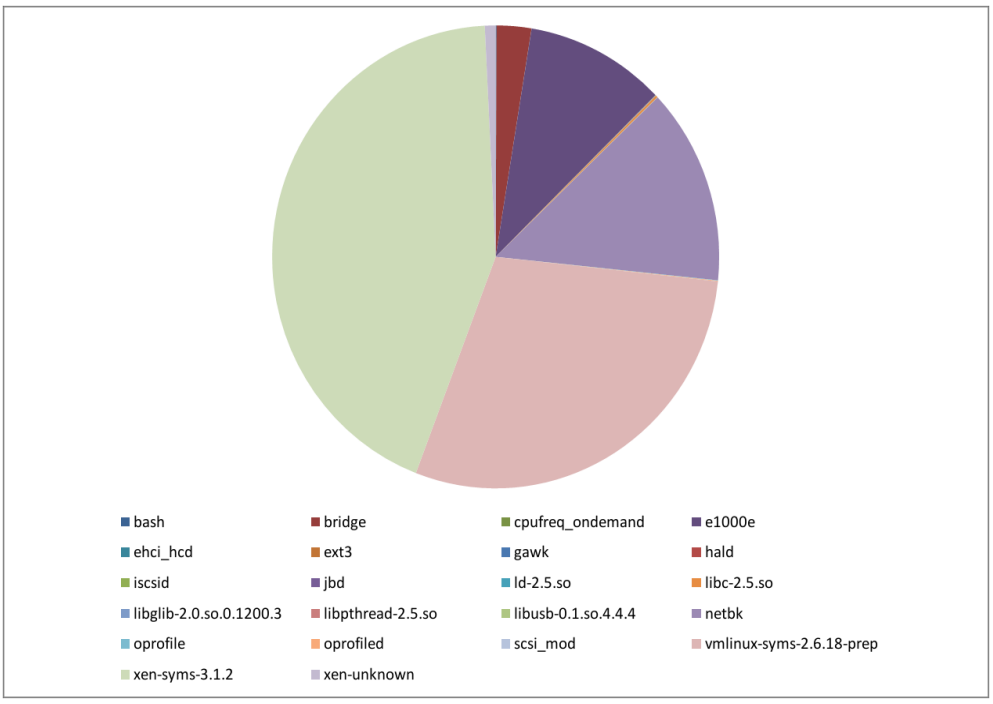


Figura 5.7: Aplicativos que geram os eventos detectados no sistema para 242 clientes do nó servidor de vídeo usando o monitor de máquinas virtuais Xen e uma única máquina virtual

A Figura 5.7 apresenta o perfil geral do programas ou aplicativos que geram os eventos detectados pela ferramenta de **OProfile**. Nela é possível observar que a maior parte dos símbolos capturados pertencem ao **Xen**, sendo desse modo considerados como sobrecarga para o processamento.

A Figura 5.8 apresenta os sete programas ou classes com maior quantidades de eventos vistos na Figura 5.6. É possível observar na Figura 5.8 que os eventos que ocorreram com maior frequência no intervalo de tempo considerado foram os eventos ligados ao gerenciamento do Xen para controle de acesso aos dispositivos de entrada/saída como o, *do_grant_table_op* e *__flush_tlb_mask*. Esse resultado explica para o sistema computacional usado nos experimentos a incapacidade do **Xen** de utilizar mais da banda passante disponível no hardware. Sendo que nos experimentos apresentados neste trabalho a banda passante total utilizada pelo **Xen** nunca foi superior a 677, 6 **Mbps** o que é equivalente a aproximadamente

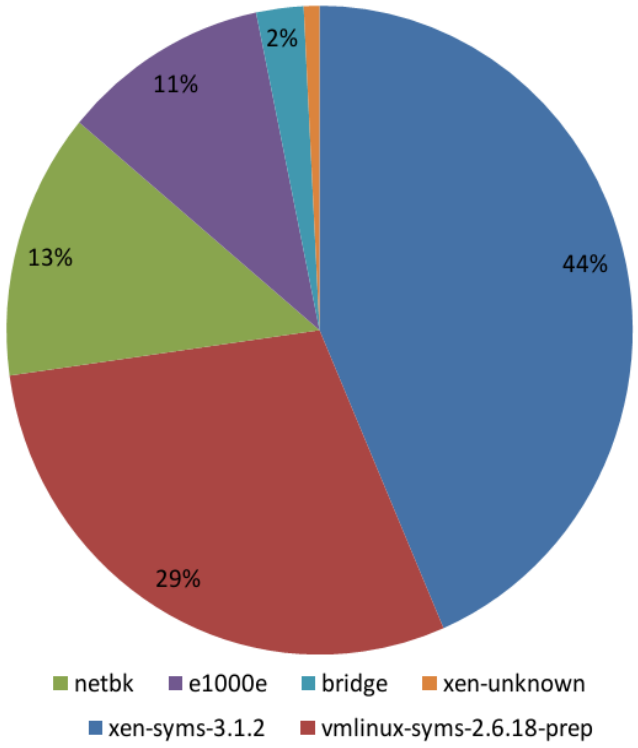


Figura 5.8: Amostragem de eventos para 242 clientes nó servidor de vídeo usando o monitor de máquinas virtuais Xen e uma única máquina virtual

76% da capacidade total disponível no *hardware*.

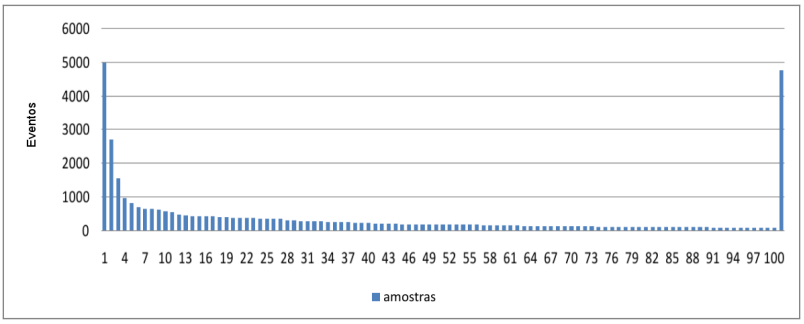


Figura 5.9: Distribuição de eventos para 242 clientes nó servidor de vídeo usando o monitor de máquinas virtuais Xen e duas máquinas virtuais

No cenário experimental representado pela Figura 5.6, foram capturados um total de 32287 eventos em um intervalo de 60 segundos, por sua vez o mesmo experimento no cenário da Figura 5.9 onde são utilizadas duas máquinas virtuais distintas cada uma conseguindo tratar 121 clientes a distribuição dos eventos

que mais ocorrem no intervalo de interesse não se altera. A única diferença foi a ocorrência de um número de eventos capturados ser 13% superior, 36493 eventos em um intervalo de 60 segundos.

### 5.2.4 Transmissão de vídeo usando a virtualização total com o KVM

Os experimentos com o monitor de máquinas virtuais KVM, busca primeiro avaliar comparativamente para um sistema de transmissão de vídeo os efeitos que a implementação da técnica de virtualização total do KVM afeta um sistema de transmissão de vídeo, e em segundo lugar, verificar como se comporta essa mesma implementação quando variamos o número de máquinas virtuais concorrentemente ativas no *hardware*.

Como descrito anteriormente na Seção 2.1.2, no KVM os domínios não privilegiados não acessam diretamente os dispositivos de entrada/saída, realizando este acesso através do **QEMU** que executa no domínio privilegiado, isso pode ser observado nos experimentos pelo fato de os eventos que mais ocorreram no sistema terem sido ligados ao aplicativo **qemu-kvm** marcados na Figura 5.10 como os eventos de classe 1. Note que a Figura 5.10 representa o experimento com apenas uma máquina virtual atendendo a todos os 242 clientes. Observe ainda, que como no cenário com o **OpenVZ**, devido a grande quantidade de classes e eventos capturados durante o intervalo de interesse dos experimentos, a Figura 5.10 fornecer a distribuição quantitativa dos mesmos.

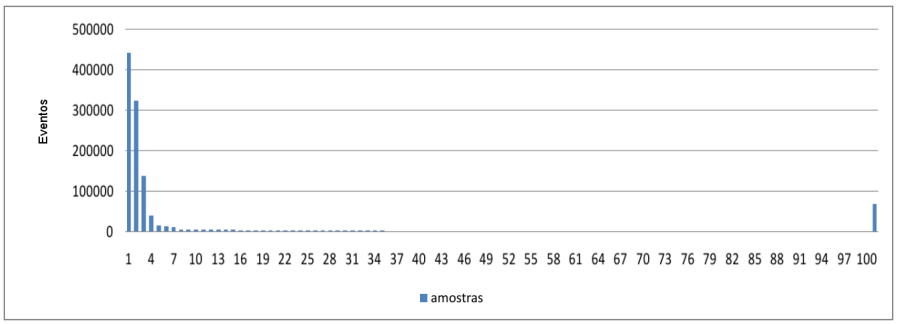


Figura 5.10: Distribuição de eventos para 242 clientes nó servidor de vídeo usando o monitor de máquinas virtuais KVM e uma única máquina virtual

Essa forma de controle ao acesso dos dispositivos imposta pela implementação de KVM faz com que a quantidade máxima de clientes suportados por uma única máquina virtual no *hardware* utilizado nunca fosse superior a 242 clientes. A Tabela 5.4 apresenta os resultados obtidos para diferentes configurações no número de máquinas virtuais para o KVM, onde na primeira coluna é o número máquinas virtuais executadas concorrentemente, a segunda coluna contendo o total de clientes servidos por cada máquina virtual e a terceira coluna apresenta apenas o total de clientes atendidos pelo nó físico, ou seja, Total de Clientes = Número de VM's * Total de Clientes por VM. Ainda sobre a Tabela 5.4, os

casos em que são apresentados o número de clientes fracionários significando que duas das máquinas virtuais receberam um cliente a mais que as demais.

Tabela 5.4: Sumário do experimento para o KVM com vídeo de taxa de 2.8 Mbps

| Número de VM's | Total de Clientes por VM | Total de Clientes no *hardware* |
|---|---|---|
| 1 | 242 | 242 |
| 2 | 141 | 282 |
| 4 | 80 | 320 |
| 8 | 40 | 320 |

Com relação à quantidade de máquinas virtuais que foram executadas concorrentemente durante os experimentos sumarizados na Tabela 5.4, verificou-se que o KVM, ao contrário do **Xen**, consegue alcançar o número máximo de clientes que o servidor consegue suportar quando são utilizadas 4 ou mais máquinas virtuais. Isso ocorre porque no KVM cada máquina virtual possuiu um processo no nível do usuário do domínio privilegiado responsável por realizar as suas operações de entrada/saída, o que possivelmente aumenta o grau de concorrência dentro do sistema e permite uma utilização melhor dos recursos de rede disponíveis.

A Figura 5.11apresenta os sete programas ou classes com maior quantidades de eventos vistos na Figura 5.10. É possível observar na Figura 5.11, que os eventos que ocorreram com maior frequência no intervalo de tempo considerado foram os eventos ligados ao gerenciamento do **QEMU-KVM** para controle ao acesso aos dispositivos de entrada/saída, eventos do processamento do kernel do Linux ligados a copia de páginas e do *driver* da interface de rede. E enquanto que com o aumento no número de máquinas virtuais foi possível utilizar a capacidade total do servidor com relação ao número de clientes atendidos concorrentemente, nos experimentos apresentados neste trabalho a banda passante utilizada por uma máquina virtual no KVM nunca foi superior a $677,6$ **Mbps** o que é equivalente a aproximadamente 76% da capacidade total disponível no *hardware*.

## 5.3 Discussão dos Resultados

Os neste capítulo foram apresentados resultados que avaliaram como diferentes técnicas de virtualização apresentadas no Capítulo 2 afetam o desempenho de um servidor Web para transmissão de vídeo. Os resultados obtidos através dos experimentos demonstraram que em termos de utilização efetiva do hardware, em relação a banda passante de rede em Mbps ou número de clientes foram atendidos pela mesma máquina física, apenas o OpenVZ conseguiu ter um desempenho comparável com o Linux sem virtualização quando foi utilizado apenas uma máquina virtual. Contudo um resultado interessante foi o KVM ter conseguido utilizar melhor o hardware quando foram utilizadas mais máquinas virtuais, chegando ao mesmo desempenho de rede do Linux sem virtualização a partir de quatro máquinas virtuais.

Sendo possível observar com base nos resultados apresentados na Seções 5.2.3 e 5.2.4, que mesmo em um cenário onde apenas um servidor físico é utilizado

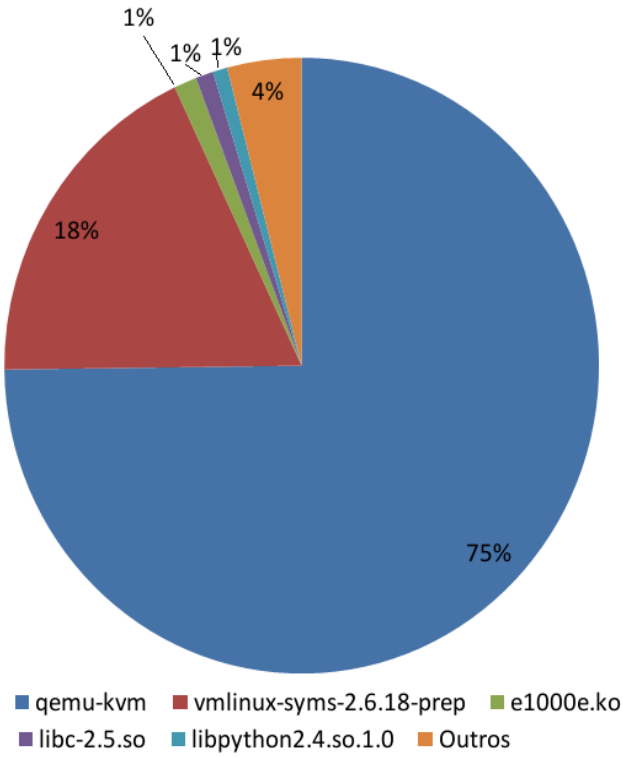


Figura 5.11: Amostragem de eventos para 242 clientes nó servidor de vídeo usando o monitor de máquinas virtuais KVM e uma única máquina virtual

uma aplicação elástica de transmissão de vídeo sob demanda consegue com o uso do KVM, quando se aumenta o número de máquinas virtuais, utilizar toda a vazão de rede disponível no servidor o que não ocorreu com o Xen, demonstrando deste modo que o uso de um serviço de vídeo consegue diferenciar por exemplo, uma arquitetura virtualziada usando Xen de uma outra que utiliza KVM. Por fim tais resultados serão utilizados como guia no desenvolvimento da aplicação de distribuição de vídeo elástica apresentada na Seção 4.1.

# Capítulo 6

# Conclusão

Este capítulo traz as considerações finais deste relatório. Os resultados experimentais demonstraram que o serviço de transmissão de vídeo sob demanda forneceu meios para avaliar diferenças nas técnicas de virtualização que aparecem no escopo deste trabalho, de modo que a continuidade do trabalho será dada a partir do uso do servidor de transmissão de vídeo escalável como ferramenta de testes de diferentes arquiteturas virtualizadas. A escolha do serviço web de transmissão de vídeo surge devido as demandas de tempo real que este tipo de aplicação têm.

## 6.1 Conclusões

Os trabalhos relacionados apresentados no Capítulo 3 concentraram-se na avaliação de desempenho das arquiteturas virtualizadas, quando o conjunto de recursos computacionais é estático. Com base nos resultados apresentados ao longo do Capítulo 5 foi demonstrado que um servidor de transmissão de vídeo sob demanda, consegue aferir quantitativamente diferenças no desempenho entre as técnicas de virtualização apresentadas na Seção 2.1. Com base nos trabalhos relacionados, estudo e experimentos realizados ao longo dos últimos dois anos é possível afirmar que um serviço de distribuição de vídeo sob demanda executando sobre uma arquitetura virtualizada sofre diferentes impactos dependendo da técnica de virtualização utilizada pela arquitetura. Como indicam os resultados experimentais das Seções 5.2.3 e 5.2.4 onde é possível observar que o uso de diferentes técnicas de virtualização em combinação com a variação no número de máquinas virtuais apresentou uma utilização melhor da banda de rede disponível no servidor para o KVM em comparação com o Xen.

# Referências Bibliográficas


[1] Lauro Luis Armondi Whately. *Uso de Virtualização na construção de serviços de streaming escaláveis*. PhD thesis, COPPE/Sistemas, UFRJ, Rio de Janeiro, RJ, Brasil, July 2008. (in Portuguese).

[2] R. Uhlig, G. Neiger, D. Rodgers, A.L. Santoni, F.C.M. Martins, A.V. Anderson, S.M. Bennett, A. Kagi, F.H. Leung, and L. Smith. Intel virtualization technology. *Computer*, 38(5):48–56, May 2005.

[3] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig. Intel virtualization technology: Hardware support for efficient processor virtualization. *Intel Technology Journal*, 10(3), Aug. 2006.

[4] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, third edition, August 2004).

[5] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, 1974.

[6] R. J. Creasy. The origin of the vm/370 time-sharing system. *IBM Journal of Research & Development*, 25(5):483–490, Sept. 1981.

[7] J. E. Smith and R. Nair. *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann, June 2005.

[8] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual machine monitor. In *OLS '07: The 2007 Ottawa Linux Symposium*, pages 225–230, jul 2007.

[9] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, Al. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *ACM Symposium on Operating Systems Principles*, pages 164–177, 2003.

[11] Ian Pratt, Keir Fraser, Steven Hand, Christian Limpach, Andrew Warfield, Dan Magenheimer, Jun Nakajima, and Asit Mallick. Xen 3.0 and the art of virtualization. In *Proceedings of Linux Symposium 2005*, July 2005.

[12] Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh. The design and implementation of zap: a system for migrating computing environments. *SIGOPS Oper. Syst. Rev.*, 36(SI):361–376, 2002.

[13] SWsoft. Virtuozzo containers 4, May 2008. http://www.parallels.com/en/products/virtuozzo/.

[14] Matt Helsley. *LXC: Linux container tools.* IBM, Fevereiro 2009.

[15] Steve Muir. The seven deadly sins of distributed systems. In *First Workshop on Real, Large Distributed Systems, WORLDS'04*, Dec. 2004. http://www.usenix.org/events/worlds04/tech/muir.html.

[16] SWsoft. *OpenVZ Users Guide*, Nov. 2005.

[17] Linux container. Web.

[18] Sukadev Bhattiprolu, Eric W. Biederman, Serge Hallyn, and Daniel Lezcano. Virtual servers and checkpoint/restart in mainstream linux. *SIGOPS Oper. Syst. Rev.*, 42:104–113, July 2008.

[19] Oren Laadan and Serge E. Hallyn. Linux-cr: Transparent application checkpoint-restart in linux. In *The 2010 Ottawa Linux Symposium, OLS '10*, jul 2010.

[20] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, University of California at Berkeley, 2009.

[21] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *IEEE 2008 Grid Computing Environments Workshop, GCE '08*. IEEE, 2008.

[22] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39:50–55, December 2008.

[23] What is software as a service (saas) ? Web, 2011.

[24] Salesforce customer relationships management (crm) system. Web, 2011.

[25] What is platform as a service (paas) ? Web, 2011.

[26] Google app engine. Web, 2011.

[27] Navraj Chohan, Chris Bunch, Sydney Pang, Chandra Krintz, Nagy Mostafa, Sunil Soman, and Rich Wolski. Appscale: Scalable and open appengine application development and deployment. In *First International Conference on Cloud Computing, CloudComp'09*. ICST, 2009.

[28] Amazon elastic compute cloud. Web, 2011.

[29] Enomalism elastic computing infrastructure. Web, 2011.

[30] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 124–131, Washington, DC, USA, 2009. IEEE Computer Society.

[31] Borja Sotomayor, Rubén S. Montero, Ignacio M. Llorente, and Ian Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13:14–22, September 2009.

[32] Openstack cloud software. Web, 2011.

[33] Eucalyptus. *Eucalyptus Administrator's Guide (2.0)*, 2011.

[34] Amazon web services - amazon simple storage service. Web, 2011.

[35] Borja Sotomayor, Rubén Santiago Montero, Ignacio M. Llorente, and Ian Foster. Capacity leasing in cloud systems using the opennebula engine. In *The First Workshop on Cloud Computing and its Applications, CCA'08*, 2008.

[36] OpenNebula.org. *Managing Hosts and Clusters 2.2*, 2011.

[37] OpenStack.org. *OpenStack Compute: Administration Manual*, Abril 2011.

[38] OpenStack.org. *OpenStack Object Storage: Administration Manual*, Maio 2011.

[39] Openstack image - glance. Web, 2011.

[40] Leonardo B. Pinho and Claudio L. Amorim. Assessing the efficiency of stream reuse techniques in p2p video-on-demand systems. *Journal of Network and Computer Applications (JNCA)*, 29(1):25–45, January 2006. ISSN: 1084-8045, Academic Press - Elsevier.

[41] Leonardo Leal Sampaio Bragato. Implementação e avaliação de um sistema de vídeo sob demanda baseado em cache cooperativa colapsada de vídeo. Master's thesis, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, Setembro 2006. (in Portuguese).

[42] Edison Ishikawa and Claudio L. Amorim. Collapsed cooperative video cache for content distribution networks. In *Proceedings of the Brazilian Simposium on Computer Networks (SBRC)*, pages 249–264, Natal, RN, Brazil, May 2003.

[43] Jeffrey P. Casazza, Michael Greenfield, and Kan Shi. Redefining server performance characterization for virtualization benchmarking. *Intel Technology Journal*, 10(03):243–252, 2006.

[44] Pradeep Padala, Xiaoyun Zhu, Zhikui Wang, Sharad Singhal, Kang G. Shin, Pradeep Padala, Xiaoyun Zhu, Zhikui Wang, Sharad Singhal, and Kang G. Shin. Performance evaluation of virtualization technologies for server consolidation. Technical report, HP Laboratories, HP, 2007.

[45] Padma Apparao, Ravi Iyer, Xiaomin Zhang, Don Newell, and Tom Adelmeyer. Characterization & analysis of a server consolidation benchmark. In *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '08, pages 21–30, New York, NY, USA, 2008. ACM.

[46] Lamia Youseff, Rich Wolski, Brent Gorda, and Chandra Krintz. Evaluating the performance impact of xen on mpi and process execution for hpc systems. In *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, VTDC '06, pages 1–, Washington, DC, USA, 2006. IEEE Computer Society.

[47] Lucas Nussbaum, Fabienne Anhalt, Olivier Mornard, and Jean-Patrick Gelas. Linux-based virtualization for hpc clusters. In *Proceedings of the Linux Symposium*, jul 2009.

[48] Donald Kossmann, Tim Kraska, and Simon Loesing. An evaluation of alternative architectures for transaction processing in the cloud. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 579–590, New York, NY, USA, 2010. ACM.

[49] Will Sobel, Shanti Subramanyam, Akara Sucharitakul, Jimmy Nguyen, Hubert Wong, Arthur Klepchukov, Sheetal Patil, Armando Fox, and David Patterson. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. In *The First Workshop on Cloud Computing and its Applications, CCA'08*, 2008.

[50] Yohei Ueda and Toshio Nakatani. Performance variations of two open-source cloud platforms. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'10)*, IISWC '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.